Institute of Visualization and Interactive Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2381

Adaptive Visualisierung koronaler Löcher

Sebastian Jähne

Course of Study:	Computer Science
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Dr. Filip Sadlo
Commenced:	Mai 09, 2012
Completed:	November 08, 2012
CR-Classification:	I.3.8 I.6.6 J.2

Abstract

Coronal holes are structures on the photosphere of the sun. They are in close relation to so-called geomagnetic storms, which have the potential of harming electronic devices in space and even on Earth. We present a method for extracting the coronal holes from magnetic field data of the Sun in an adaptive refinement manner. In contrast to previous methods our algorithm produces vector graphics representation instead of a texture which would show coarsely approximated coronal holes only. It takes advantage of the fact that the magnetic field of the Sun can be topologically divided into two parts which define the coronal hole boundaries.

Contents

List of Figures

List of Algorithms

1. Introduction

Overview

This thesis is structured in the following manner:

Chapter 1 – Introduction: A brief description of the method.

Chapter 2 – Related Work: Other works on this field.

Chapter 3 – Data Model: The used data model.

Chapter 4 – Extraction of Coronal Hole Boundaries: A detailed description of the method.

Chapter 6 – Results: Runtime estimation, measurements and future works.

Chapter 5 – Implementation: Implementation details.

Chapter 7 – Algorithms: Pseudocode for the most important algorithms.

Chapter 8 – Conclusion: Conclusion

Before we start with the topic of this thesis, we will give a short description of the structure of the Sun. Its radius r_s (i.e., the radius of the photosphere) is around $6.96342 \cdot 10^8 m$. The Sun can be divided into 6 parts (see figure 1.1):

- **Core**: It lies in the center of the Sun and is its hottest part. Its temperature is roughly $15.7 \cdot 10^6 K$ and its diameter is considered to be $\frac{1}{4}$ of the Sun's radius r_s. Most of the energy produced by nuclear fusion of hydrogen nuclei to helium nuclei originates here.
- **Radiative Zone**: Inside this zone, energy is transported through radiation. The matter however is still very dense, resulting in very slow transportation of energy. Gamma rays emitted from the core take around 171000 years for crossing the radiative zone.
- **Convective Zone**: As opposed to the radiative zone, the convective zone transports energy primary through the process of convection. Like air in the atmosphere of the earth, the hot particles move away from the center of the Sun and the colder particles fall back down.
- **Photosphere**: The photosphere is the first layer of the Sun's atmosphere. Its name comes from the fact that most of the light emitted by the Sun originates here. Its temparature lies between $4500K$ and $6000K$.

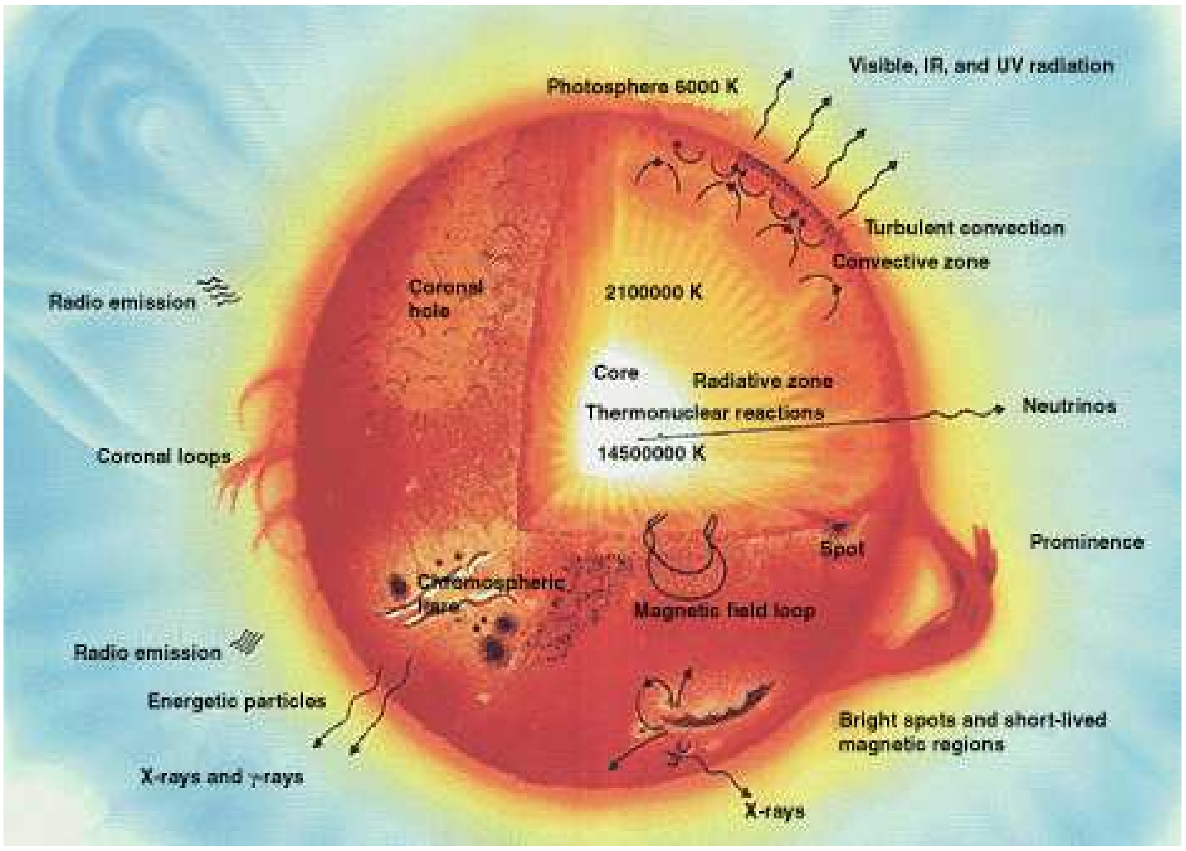

Figure 1.1.: The structure of the Sun.[1]

- **Chromosphere**: It is the second layer of the atmosphere of the Sun and its extension is 2000 kilometer. It holds an interesting feature which is still not fully understood: in contrast to the other parts of the Sun, the temparature rises from $4400K$ up to $25000K$ with increasing distance from the photosphere. It is believed that this effect can be explained by a phenomenon called magnetic reconnection [Cha07]: through the process of induction, magnetic energy is transformed into currents, which collapse and produce kinetic energy.

- **Corona**: The Sun's corona is the structure which is of interest for us. It extends far into the interplanetary space and is very hot. The average temparature is around $3 \cdot 10^6 K$. It is still unknown how the corona gets heated up, but it seems likely that the same effect that is responsible for heating up the chromosphere is responsible for the rise in temparature. Apart from the magnetic reconnection theory exists another theory called wave heating theory [Alf47]: It states that the corona is heated up by different waves originating from the Sun. See also [MZO+99] for a magnetohydrodynamic context of this matter.

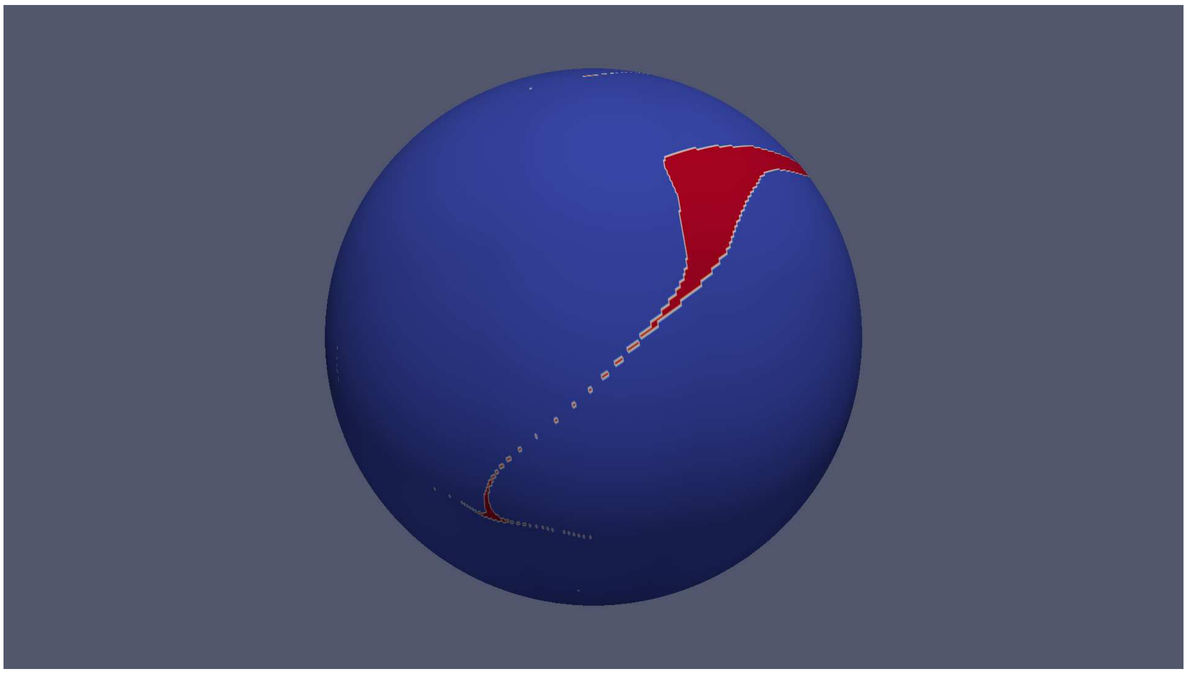

Figure 1.2.: Texture based method for rendering the coronal holes.

Nowadays more and more electric and electronic devices exist, therefore one natural phenomenon gains major importance: geomagnetic storms. They emerge from the Sun due to massive coronal mass ejection and can harm electric circuits with their high energy particles.

The Sun has, like the Earth, an electromagnetic field due to the motion of conductive material in its center. But this field is not static, it changes over time and is also not evenly distributed over the whole corona, i.e., the atmosphere of the Sun. There are regions where the field is particulary weak so that plasma can escape the corona. These areas, which are called coronal holes, are characterized by so-called open field lines, meaning that a fieldline starting at such a position on the Sun's photosphere does transcend a specific radius r_c, defined below.

A common approach is to starting fieldlines on the surface of the Sun and checking if they cross this radius, one can obtain a texture that indicates where these holes are[2]. It is an easy to implement method but has several drawbacks, which are all linked to the fact that the surface is discretized in this approach. An interesting observation of coronal holes is that they are seemingly not always contained areas, but seem to create corridors which link them together. The motivation for using our method instead of the texture based method is to visualize these structures.

[1]http://imagine.gsfc.nasa.gov/docs/science/know_l2/sun_parts.html
[2]http://www.predsci.com/hmi/home.php

This thesis presents a different approach by adaptively starting fieldlines on structures of the corona rather than on the surface, which are also characterized by the magnetic field and motivated by the so-called boundary switch curves [WTHS04]. In the next step the computed boundaries of the coronal holes are refined further, until the resolution of the borders is sufficient for revealing very thin features like passages between them.

For the methods used in this thesis, only the magnetic field data is needed. There is no need for additional data like the current or vorticity of the plasma, since coronal holes are defined as structures arising from the magnetic field.

The first step in extracting coronal holes is to define what exactly a coronal hole is. For this thesis, the following definition is the basis:

Definition 1. Coronal holes are regions of low-density plasma on the Sun that have magnetic fields that open freely into interplanetary space.[3]

Let $\mathbf{B} : \mathbb{R}^3 \to \mathbb{R}^3, (x, y, z) \mapsto \mathbf{B}(x, y, z)$ be the magnetic field. For determining if the magnetic field is open at a respective point, one must integrate therefrom along the magnetic field. This task is done by constructing so-called streamlines. A Streamline $L(s)$ can be described as an ordinary differential equation: $\frac{dL(s)}{ds} = \mathbf{B}(L(s))$, which states that at every point of the streamline the tangent must be always parallel to the underlying field at the same position.

According to definition 1, the field must open freely into interplanetary space. The dual to this statement is that all regions where streamlines originate and at some point and retract to the Sun's surface are regions which are not a part of a coronal hole.

Since the domain of the data is limited with respect to radius r, an arbitrary radius r_c from the origin of the Sun must be defined at which a streamline is considered to extend to free space and hence open. Usually this radius is set to 2.5 times the radius of the Sun r_s, e.g., $r_c = 2.5 \cdot 6.96342 \cdot 10^8 m = 17.40855 \cdot 10^8 m$.

Starting streamlines at the surface, and checking if the radius is overstepped, and marking these seedpoints as parts of coronal holes, is the way coronal holes are usually extracted. But since streamlines divide the field in a topological manner in parts where the flow is separated from another, one can reverse this approach and start streamlines at specific points outside the Sun and consider everything below these streamlines as contained areas where no plasma escapes the Sun. Everything that is not encapsulated by these streamlines is a candidate for an open field structure. The question that arises is how one can set the seedpoints so that all regions, which are not part of this open field structure, are contained by these densely seeded streamlines.

The first approach for doing this was to extract countour lines with $\mathbf{B}_r = 0$ at the radius r_c with $\mathbf{B}_r$ being the radial component of the magnetic field $\mathbf{B}$. But this approach had major

[3] https://www.cfa.harvard.edu/~scranmer/Preprints/eaaa_holes.pdf

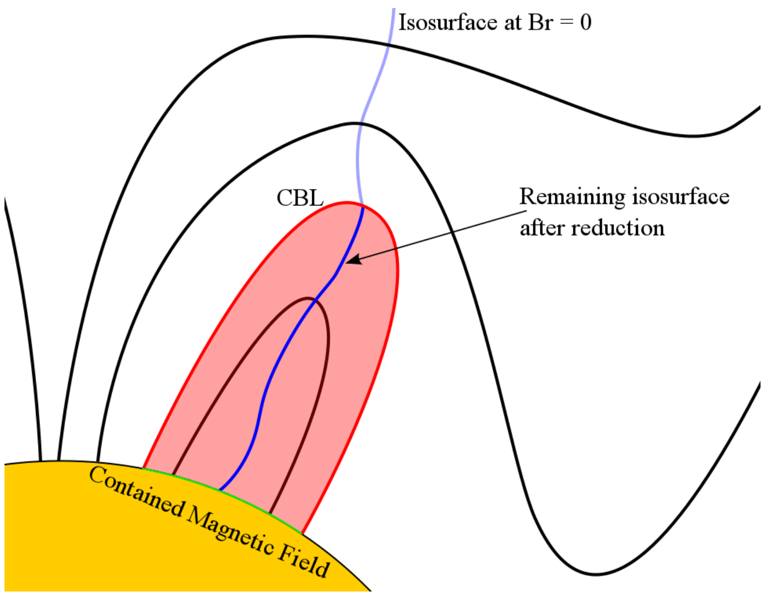

Figure 1.3.: Relation between coronal boundary lines and the isosurface at $\mathbf{B}_r = 0$.

flaws. One of them was the fact that not all of the streamlines which were started on these contour lines were retracting to the Sun, i.e., either one end or both streamlines started at the seed might extend away from the Sun, which yielded into the segmentation of these contour lines: Only segments where all started streamlines retract to the Sun were considered to be of importance (see figure 6.3). This, on the other hand, leaded to disrupted coronal hole contours which had to be closed in some way. An approach was investigated to use critical points in the magnetic field and start streamlines from there to fill the gaps, but turned out to be futile.

To conquer this problem, the contour lines were replaced by a subset of isosurfaces: First, the isosurfaces of $\mathbf{B}_r = 0$ were constructed. These isosurfaces were then reduced by the condition that only parts of the isosurface where streamlines started in both forward and reverse direction are retracing to the Sun are kept. All other parts of the isosurfaces are dismissed. Hence, streamlines started from the remaining isosurface subsets will always retract to the Sun. M_{CMF} is the set consisting of all these lines.

We now define a set of streamlines, called coronal boundary lines (CBLs), which are illustrated in figure 1.3.

Definition 2. Coronal boundary lines (CBLs) are the lines which divide the corona in two distinct regions: A region which contains only closed field lines and a region which contains only open field lines.
A line l is a CBL iff it is in ∂M_{CBL}, the boundary of M_{CBL}

Iff $l \notin M_{CBL}$, at least one end of the streamline l will escape into the open space.

It might be tempting to replace the definition of enclosed points by the convex hull, but this is not valid: A CBL is not necessarily convex. In fact, we observed that many CBLs are not convex. In our case, about 50% of the CBLs were not convex. With the CBLs, it is now possible to partition the magnetic field into two parts: A subset where the field lines are contained, and another part where the field lines are open. The intersections of streamlines belonging to a CBL with the sphere of the radius r_s define a set of points which represents the boundaries of the coronal holes. If connected correctly, these points should define curves that provide a good representation of the coronal holes.

However, in our discretized approach the CBL consist only of streamlines started at the vertices of the isosurface $\mathbf{B}_r = 0$, hence these line segments on the photosphere can become too long. Additionaly, the extracted coronal hole boundaries tend to form sawtooth patterns (figure 1.4) due to the fact that the mesh boundaries are very coarse and small changes of the seeding point positions lead to major abberation. This effect gets worse with increasing distance of the seeding points to the Sun. Because of this, there is a need to adaptively refine the CBLs where the corresponding line segments on the Sun's surface are too long. A line segment is considered to be too long if its length exceeds the arc length l_s of two neighboring points of the original sampling grid at radius r_s. The refinement is done by simply subdividing the 3D cell which contains the CBL segment and applying the marching cubes algorithm to these subdivided cells. The extracted isosurface needs to be trimmed again in terms of retracting streamlines. This process is repeated until all line segments of the coronal hole underrun l_s.

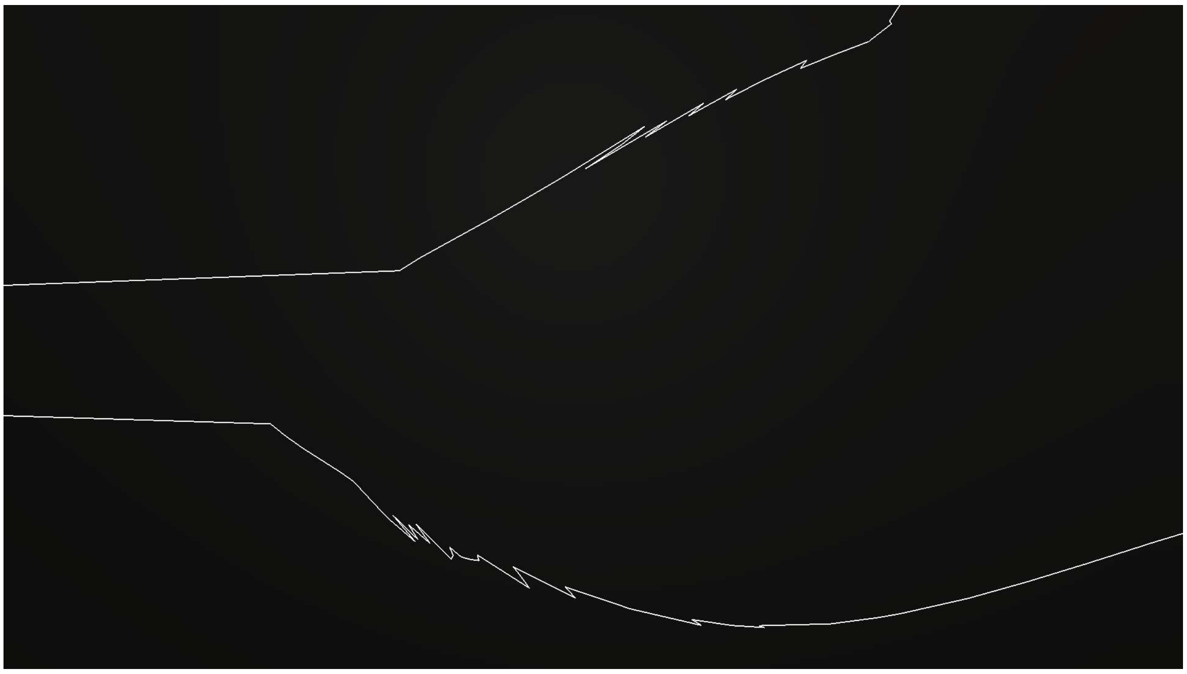

Figure 1.4.: Sawtooth pattern of the extracted coronal hole boundaries which is a result from the coarse representation of the $\mathbf{B}_r = 0$ isosurface boundaries.

2. Related Work

There have been only very few attempts to actually visualize the coronal holes. Especially in terms of 3D visualization, there are no works available until now.

Machado et al. investigated flow visualization of the magnetic field in 3D, concentrating on coronal loops and plasma convection [GFT+12]. Madjarska et al. worked on analyzing coronal hole boundary evolution on small scales [MW09]. Cranmer described methods of measuring the coronal holes and their physical properties [Cra09]. Scholl and Habbal automatically detected and classified coronal holes by combining ultraviolet images and magnetograms [SH09]. Banerjee et al. witnessed propagation of MHD waves in coronal holes [BGT11]. Krieger and Timothy identified coronal holes as sources of high velocity solar winds [KTR73]. [MC02]. Yang et al. analyzed how the magnetic field of the sun influences it's atmosphere [YZJ+09]. Zhang et al. investigated changes of surface properties inside coronal holes during solar maxima [ZWS+03]. Hearn described differences between quiet regions of the corona and coronal holes [Hea77].

3. Data Model

3.1. MHD Data

This thesis operates solely on data files derived from the Helioseismic and Magnetic Imager Solar Dynamics Observatory[1]. In this context, only the files which contain the Sun's 3D magnetic field are of interest.

These files are stored in a specific manner, namely the Hierarchical Data Format (HDF)[2]. Currently, they are only available in version 4, and not in the new version 5.

There are three separate files for the magnetic field: bp*.hdf, bt*.hdf, and br*.hdf. These contain the phi, theta and radial component of the magnetic field. Because it is numerically advantageous for the simulation, the computed magnetic field is stored in a special way, a staggered grid. Before any computation can take place, it has to be converted to a curvilinear grid for our purpose.

3.2. Staggered Data to Curvilinear Grid

The staggered grid representation provided by the HDF files can be seen in figure 3.1. The magnetic field data lies on the edges of the staggered grid, denoted as $\mathbf{B}_\phi(\square)$, $\mathbf{B}_\theta(\blacktriangle)$ and $\mathbf{B}_\mathbf{r}(\blacklozenge)$. In order to construct a curvilinear grid, interpolation is needed. In the case of the magnetic field, simple linear interpolation along $\hat{\phi}$, $\hat{\theta}$ and $\hat{r}$ is the method of choice.

See section 7.4 for the complete algorithm.

3.3. Physical and Computational Space

Since the sun can be seen as a giant spherical object, it is possible to express the transformation from physical to computational space and vice versa analytically so there is no need for algorithms like stencil walk to provide point location.

[1]http://www.predsci.com/hmi/home.php
[2]http://www.hdfgroup.org/
[3]http://www.predsci.com/mas/userguide/MAS-User-Guide.pdf

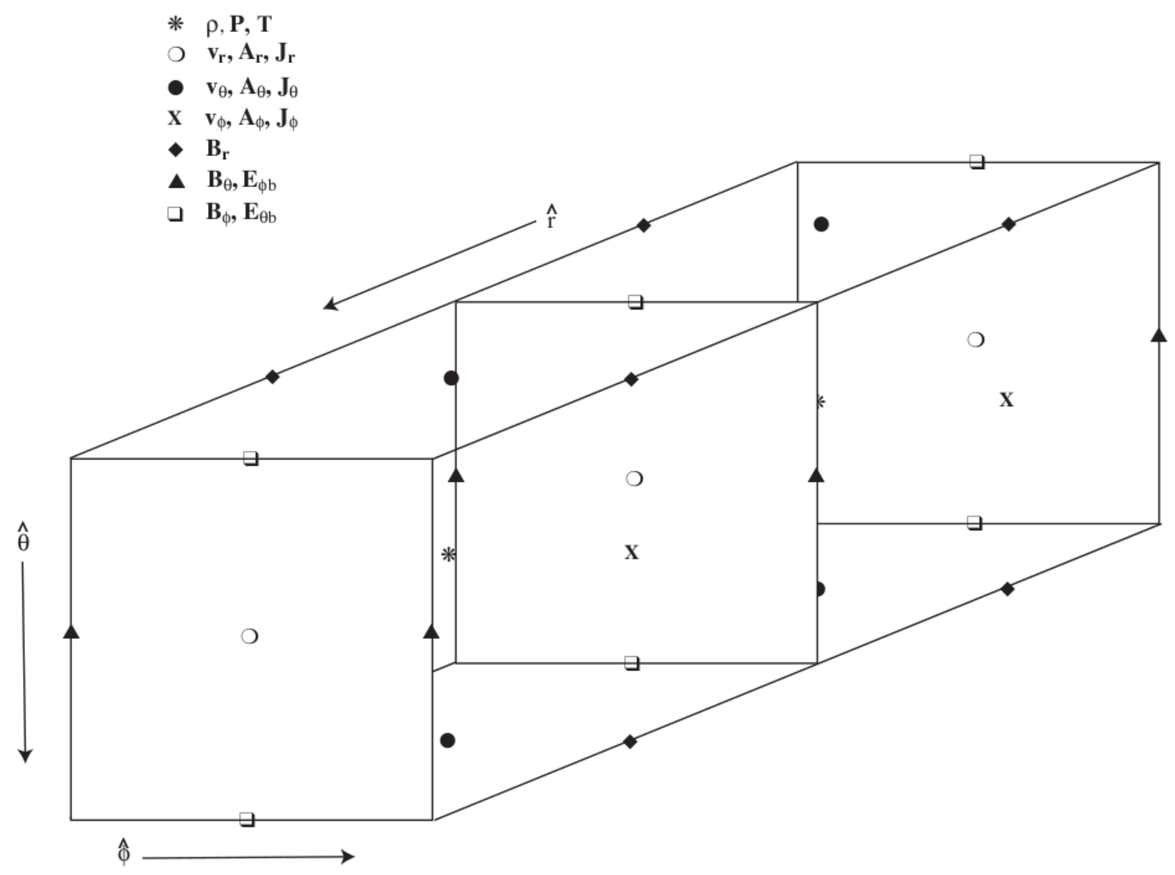

Figure 3.1.: The staggered grid representation as provided by the HDF files. $\mathbf{B}_\phi(\square)$, $\mathbf{B}_\theta(\blacktriangle)$ and $\mathbf{B}_\mathbf{r}(\blacklozenge)$ represent the corresponding components of the magnetic field.[3]

Given: A point $\vec{p}_p$ in physical coordinates.
Find: The corresponding point $\vec{p}_c$ in computational space.

The spherical coordinate system is defined as follow with $\vec{p}_C$ representing a point in computational space and $\vec{b}_C$ the magnetic vector in computational space, as provided by the MAS simulations:

$$\vec{p}_C = \begin{pmatrix} r \\ \theta \\ \varphi \end{pmatrix} \qquad \vec{b}_C = \begin{pmatrix} b_r \\ b_\theta \\ b_\varphi \end{pmatrix}$$

Given ϕ, θ and r, obtaining a position in physical space requires the following functions ψ:

$$\psi_x(\varphi,\theta,r) = r \cdot \sin(\theta) \cdot \cos(\varphi)$$

$$\psi_y(\varphi,\theta,r) = r \cdot \sin(\theta) \cdot \sin(\varphi) \qquad \Psi(\varphi,\theta,r) = \begin{pmatrix} \psi_x(\varphi,\theta,r) \\ \psi_y(\varphi,\theta,r) \\ \psi_z(\varphi,\theta,r) \end{pmatrix}$$

$$\psi_z(\varphi,\theta,r) = r \cdot \cos(\theta)$$

For the transformation of the magnetic field into the spherical coordinate system, one needs the Jacobian of Ψ.

$$\nabla\Psi(\varphi,\theta,r) = \begin{pmatrix} \frac{\partial\psi_x(\varphi,\theta,r)}{\partial\varphi} & \frac{\partial\psi_x(r,\theta,\varphi)}{\partial\theta} & \frac{\partial\psi_x(\varphi,\theta,r)}{\partial r} \\ \frac{\partial\psi_y(\varphi,\theta,r)}{\partial\varphi} & \frac{\partial\psi_y(r,\theta,\varphi)}{\partial\theta} & \frac{\partial\psi_y(\varphi,\theta,r)}{\partial r} \\ \frac{\partial\psi_z(\varphi,\theta,r)}{\partial\varphi} & \frac{\partial\psi_z(r,\theta,\varphi)}{\partial\theta} & \frac{\partial\psi_z(\varphi,\theta,r)}{\partial r} \end{pmatrix}$$

$$= \begin{pmatrix} -r \cdot \sin(\theta) \cdot \sin(\varphi) & r \cdot \cos(\theta) \cdot \cos(\varphi) & \sin(\theta) \cdot \cos(\varphi) \\ r \cdot \sin(\theta) \cdot \cos(\varphi) & r \cdot \cos(\theta) \cdot \sin(\varphi) & \sin(\theta) \cdot \sin(\varphi) \\ 0 & -r \cdot \sin(\theta) & \cos(\theta) \end{pmatrix}$$

$$(\nabla\Psi(\varphi,\theta,r))^{-1} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \cdot \sin(\varphi) & \cos(\varphi) \cdot \sin(\theta) \\ -\frac{\sin(\theta)}{r} & \frac{\cos(\theta)\cdot\sin(\varphi)}{r} & \frac{\cos(\theta)\cdot\cos(\varphi)}{r} \\ 0 & \frac{\cos(\varphi)\cdot\frac{1}{\sin(\theta)}}{r} & -\frac{\frac{1}{\sin(\theta)}\cdot\sin(\varphi)}{r} \end{pmatrix}$$

Putting everything together leads to the following equations:

$$\vec{p}_p = \Psi(\vec{p}_{c_\varphi}, \vec{p}_{c_\theta}, \vec{p}_{c_r}) \qquad \vec{b}_p = \nabla\Psi(\vec{p}_{c_\varphi}, \vec{p}_{c_\theta}, \vec{p}_{c_r}) \cdot \vec{b}_c$$

Jacobian of a vector field V:

$$\nabla V(r,\theta,\varphi) = \begin{pmatrix} \frac{\partial v_x(r,\theta,\varphi)}{\partial r} & \frac{\partial v_x(r,\theta,\varphi)}{\partial\theta} & \frac{\partial v_x(r,\theta,\varphi)}{\partial\varphi} \\ \frac{\partial v_y(r,\theta,\varphi)}{\partial r} & \frac{\partial v_y(r,\theta,\varphi)}{\partial\theta} & \frac{\partial v_y(r,\theta,\varphi)}{\partial\varphi} \\ \frac{\partial v_z(r,\theta,\varphi)}{\partial r} & \frac{\partial v_z(r,\theta,\varphi)}{\partial\theta} & \frac{\partial v_z(r,\theta,\varphi)}{\partial\varphi} \end{pmatrix}$$

Central differences:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \text{ leads to}$$

$$\nabla V(r,\theta,\varphi) = \begin{pmatrix} \frac{v_x(r+1,\theta,\varphi)-v_x(r-1,\theta,\varphi)}{2h} & \frac{v_x(r,\theta+1,\varphi)-v_x(r,\theta-1,\varphi)}{2h} & \frac{v_x(r,\theta,\varphi+1)-v_x(r,\theta,\varphi-1)}{2h} \\ \frac{v_y(r+1,\theta,\varphi)-v_y(r-1,\theta,\varphi)}{2h} & \frac{v_y(r,\theta+1,\varphi)-v_y(r,\theta-1,\varphi)}{2h} & \frac{v_y(r,\theta,\varphi+1)-v_y(r,\theta,\varphi-1)}{2h} \\ \frac{v_z(r+1,\theta,\varphi)-v_z(r-1,\theta,\varphi)}{2h} & \frac{v_z(r,\theta+1,\varphi)-v_z(r,\theta-1,\varphi)}{2h} & \frac{v_z(r,\theta,\varphi+1)-v_z(r,\theta,\varphi-1)}{2h} \end{pmatrix}$$

4. Extraction of Coronal Hole Boundaries

Our extraction of coronal hole boundaries is a multistage process. The first step is to create an isosurface for $\mathbf{B}_r = 0$. This isosurface has to be trimmed in order to obtain seeding points for the CBLs. The constructed CBLs are then intersected with a sphere of radius r_p, and the points resulting from this intersection are connected to produce the coronal hole boundary representation. These are then further refined to give a better approximation of the coronal hole boundaries.

4.1. Streamlines

Streamlines, also known as fieldlines, are used to visualize flow fields. They are characterized by the property that, at every point of the streamline, their tangent must be equal to the field direction at that point. Mathematically speaking, streamlines are first-order ordinary differential equations of the form

$$\frac{dL(s)}{ds} = \mathbf{B}(L(s))$$

with $L(s)$ being the streamline and the $\mathbf{B}$ being the vector field. This differential equation is, in our case, numerically solved using Runge–Kutta scheme. The simplest Runge-Kutta scheme is the Euler method, but a small step-size is a requirement for this method to work correctly since it is a first-order integration scheme. It is often used for computations on GPUs since many small steps are usually no problem here. But instead of using the Euler method, we implemented the fourth-order Runge-Kutta method on the GPU to ease the tedious process of choosing the right stepsize. See algorithm 7.1 for the pseudocode.

4.2. Isosurfaces

In order to generate the seedpoints for the streamlines, an isosurface is needed. There are several approaches to get an isosurface, but the best known and mostly used one is of course the marching cubes algorithm. The marching cubes algorithm visits every vertex of the dataset and marks them according to their relation to the given isovalue. With this labeling it is possible to construct a triangle mesh representation of the isosurfaces. See [LC87] for further details.

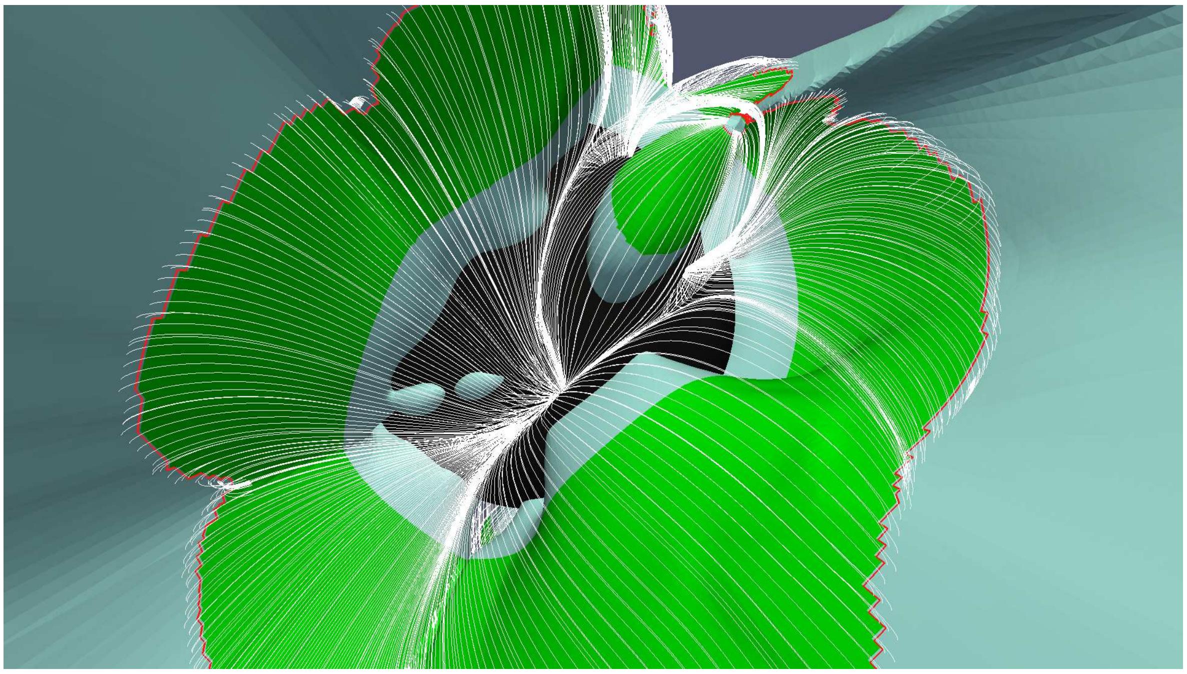

Figure 4.1.: Scheme of the generated data structures: The isosurface (light blue) at $\mathbf{B}_r = 0$ and the reduced isosurface (green). The streamlines are elements of M_{CBL}. The red line indicate the mesh boundaries

4.3. Rejecting Parts of the Isosurfaces

The isosurface alone is not sufficient for obtaining the seedpoints of the coronal holes. It has to be reduced in a specific manner. For every vertex of the isosurface, a streamline is started in forward and backward direction. If both streamlines terminate on the surface of the sun, the vertex is not deleted. If not, the vertex and its adjacent triangles are removed. What remains are triangles which are contained by the magnetic field (figure 4.2), i.e., any streamline started from a vertex of this surface will hit the photosphere outside of a coronal hole.

4.4. Mesh Boundary Extraction

Boundary extraction of the remaining meshes can be easily achieved by hash tables. We use two hash maps, one that allows multiple key-value pairs, denoted as H_A, and one which only allows distinct key-value pair, H_E, with the edges being both the key and the value. For every triangle, its edges are inserted into H_E if they are not in H_A, otherwise, they are removed from H_E. After that, they are inserted into H_A. H_E then contains all edges which belong

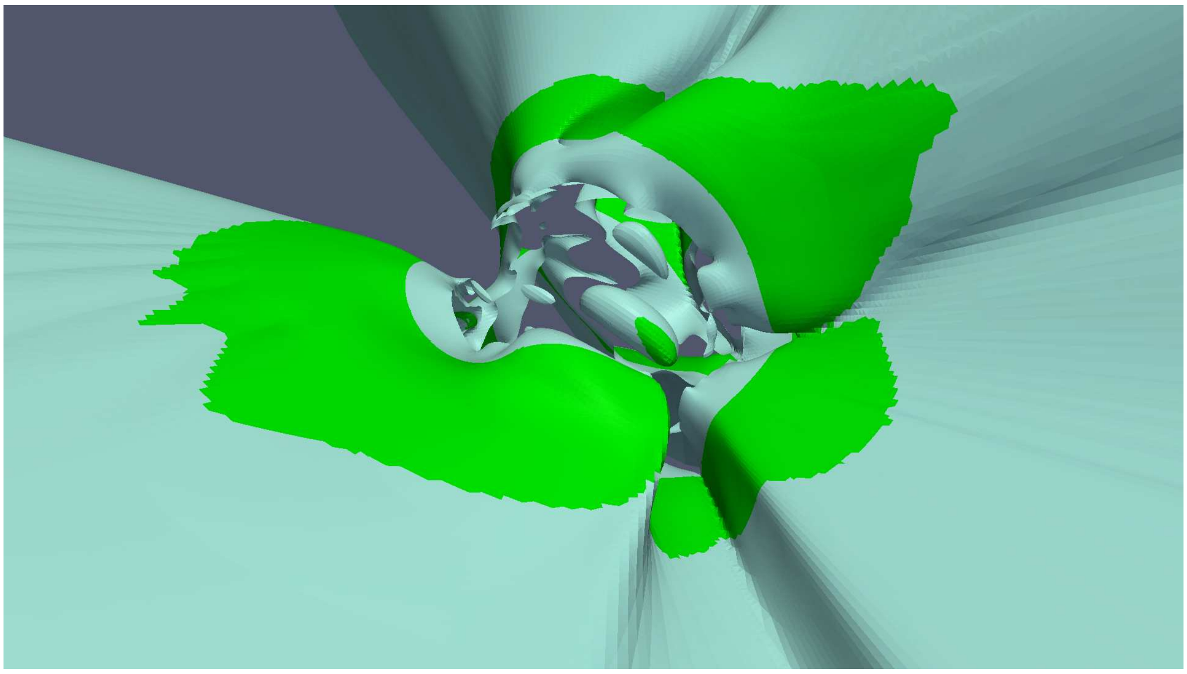

Figure 4.2.: Isosurface rejection sampling. The isosurface (light blue) at $\mathbf{B}_r = 0$ and the reduced isosurface (green).

only to one triangle, forming the required edges e_i. These edges are then concatenated to the boundary curves of the isosurface mesh parts.

4.5. Creating the Coronal Hole Boundaries

With these edges it is now possible to get the CBLs. For every vertex of each edge, two streamlines are created. The first is created integrating in forward direction of $\mathbf{B}$, the second integrating backward. These streamlines, which are not the actual CBL, but get very close to them, are then cut at the photosphere. This radius should not be too small since streamlines tend to aberrate the closer they get to the Sun's surface due to inaccuracies of the MAS simulation close to the photosphere and because many critical points exist there. The cut is done by simply walking along the streamlines and checking if the chosen radius r_p, in our case $1.2 \cdot r_s$, is underrun. r_p should be as near as possible to the photosphere of the sun, but with the MHD data supplied, at least a radius of $1.2 \cdot r_s$ should be chosen, otherwise it might not be possible to adaptively refine the coronal hole boundaries due to potential numerically issues.

After the streamlines are cut, these cuts have to be connected. Since we start the streamlines ordered along the boundaries, we can simply append the points in the same order to obtain the polylines. These lines represent the first approximation of the coronal hole boundaries.

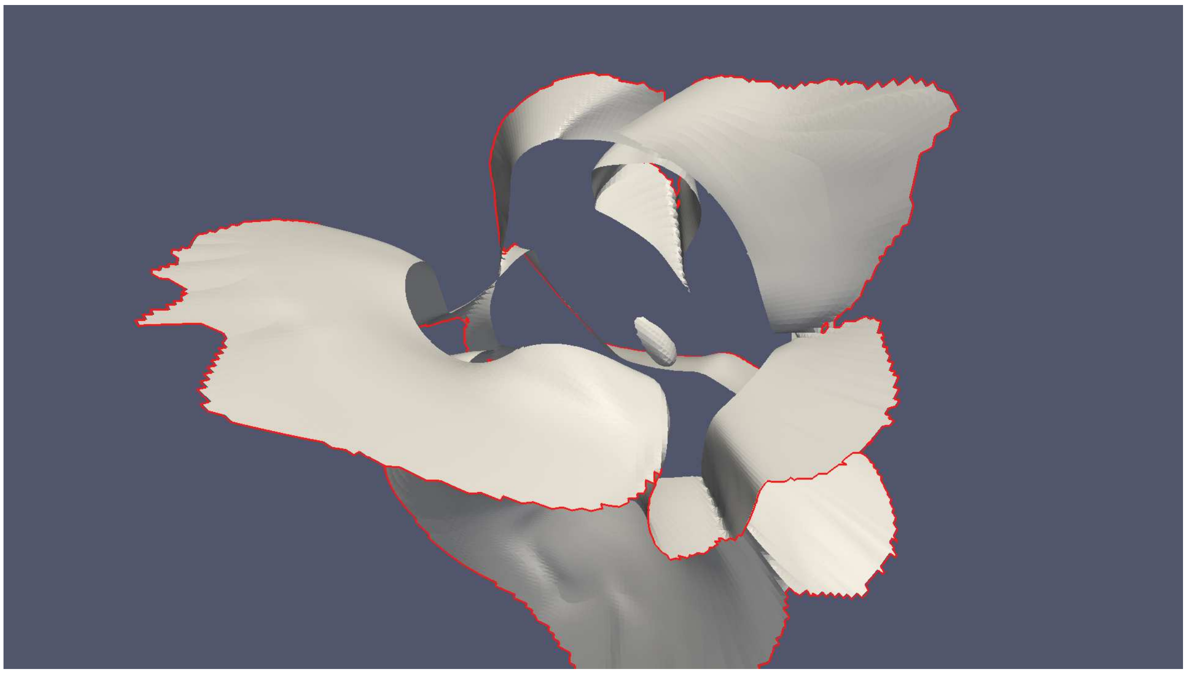

Figure 4.3.: Mesh boundary extraction.

It has to be noted that since the streamlines of **B** basically converge as they approach the comparably small photosphere, the distances between the obtained points is typically much smaller than the edge lengths where the streamlines were started from. This was one of the reasons for choosing the presented sampling approach. Furthermore, the structure of the **B** field is much more complex close to the Sun, hence starting the streamlines at some distance from the Sun provides a better sampling.

4.6. Adaptive Refinement

Some of the segments of the coronal hole boundaries are typically too long due to the diverging trajectories on **B**. This happens when a segment is, e.g., near a critical point and the streamlines are repelled from there. A segment is considered to be too long if its length is greater than the arc length defined by the resolution of the grid at the photosphere or a user defined value. In our case, a segment should be not longer than

$$\frac{\alpha\pi r}{180} = \frac{\frac{360}{\max(grid.resolution(\phi),grid.resolution(\theta))}\pi \cdot 1}{180} = \frac{\frac{360}{\max(256,150)}\pi \cdot 1}{180} \approx 0.024543693 = l_{max}$$

.

If a segment at the photpsphere is n times longer than l_{max}, with $n > 1.0$, then it is further refined through new streamlines. Since the segment is n times too long, at least $n-1$ additional

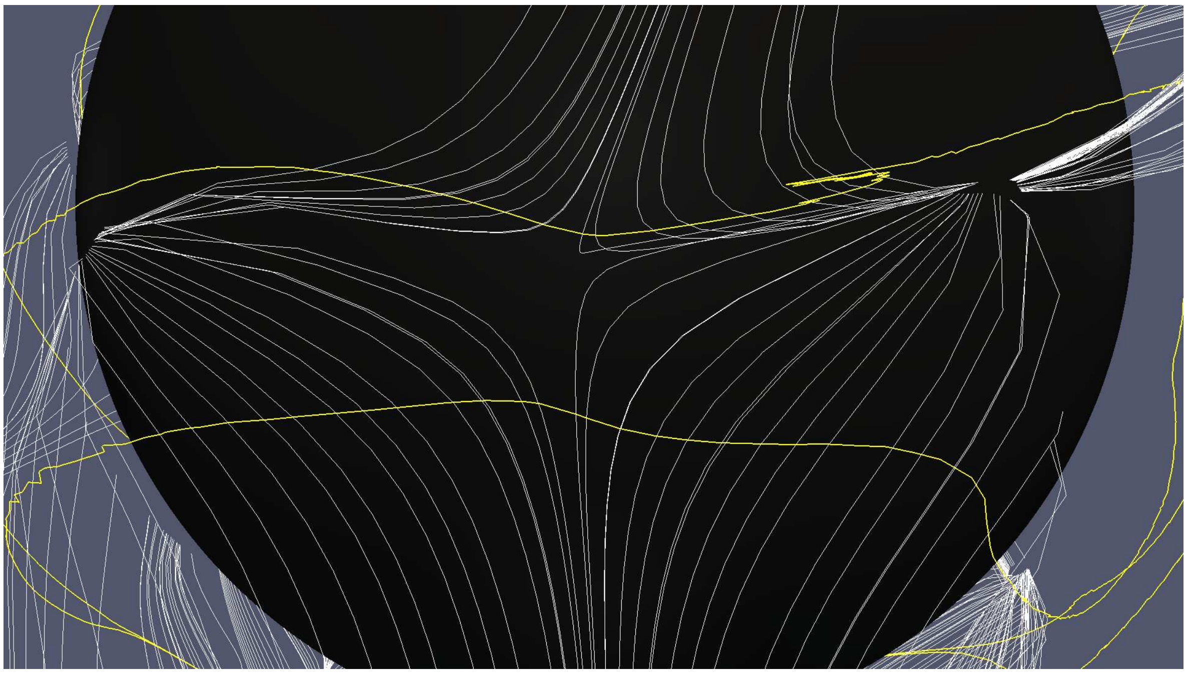

Figure 4.4.: Intersecting the CBLs with a sphere of radius $r_p = 1.2$. The yellow lines represent the coronal boundary lines.

streamlines have to be created. These streamlines have to be started from the same edge as the streamlines which belong to the segment which has to be refined. The corresponding segment on the edge can be either lineary divided into $n-1$ new segments, or even better through subdividing the containing cell into 8^n cells, performing marching cubes extraction of the $\mathbf{B}_r = 0$ isosurface on the subdivided cells and repeating the process of removing parts and detecting the boundaries. This new isosurface boundary defines new seeding points, and the streamlines started from them provide a better approximation of the boundaries of the coronal hole if intersected with a sphere of radius r_p. The whole process is repeated until no segment with length $> l_{max}$ remains or no further progress with this method is possible. This can happen due to numerically problems, e.g., if $l_m ax$ becomes very small. Another problem is that it might happen that the integration direction has to be changed for a specific seeding point, since the tangent of the streamline at the isosurface boundary is very flat. This is forcing us to always start two streamlines for every seeding point, with one streamline s_1 being integrated forward, the other streamline s_2 being integrated backward, and choosing the right one for the new point on the coronal hole boundary. This is done by computing the distances between the intersection points p_1 and p_2 of s_1 and s_2 with a sphere of radius r_p, and assessing the distances between the intersected points p_l and p_r of neighbouring streamlines s_l and s_r and choosing the point p_1 if $\sqrt{p_l^2 + p_1^2} + \sqrt{p_r^2 + p_1^2} < \sqrt{p_l^2 + p_2^2} + \sqrt{p_r^2 + p_2^2}$, otherwise point p_2 is chosen.

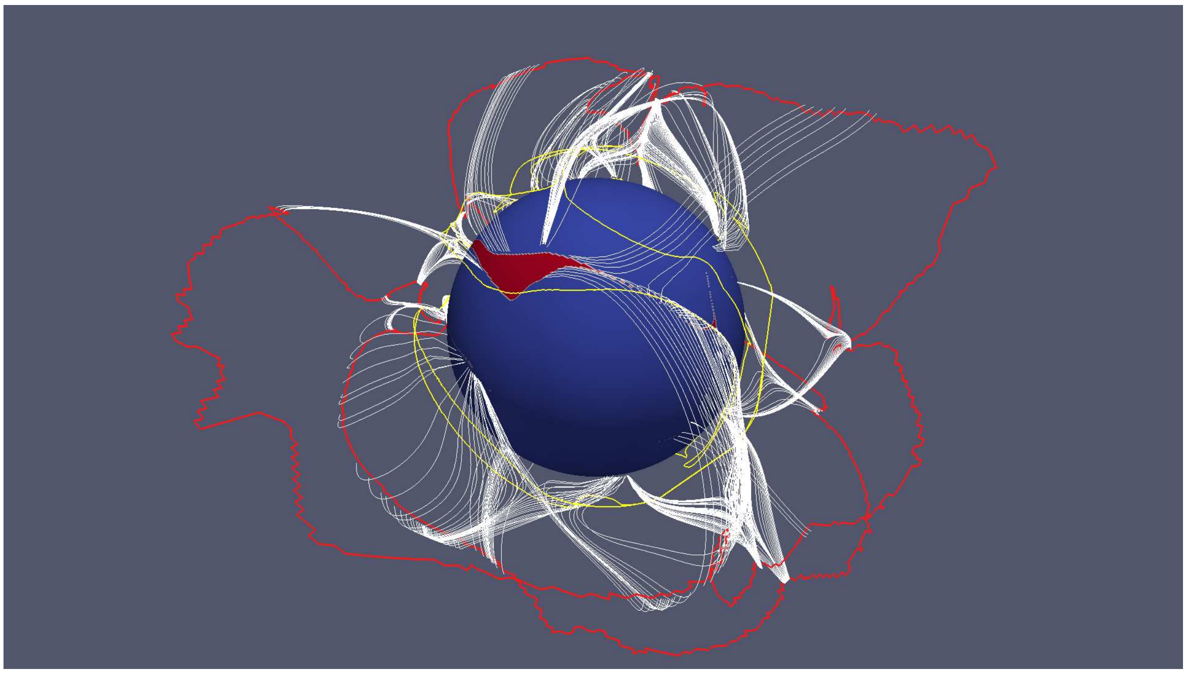

Figure 4.5.: Streamlines started from the mesh boundaries due to adaptive refinement.

Using this method, there might be false positives or negatives, but these did not occure with our data. A better way for deciding which streamline is the right one is by integrating only one step forward and backward and checking if the direction of these lines abruptly changes.

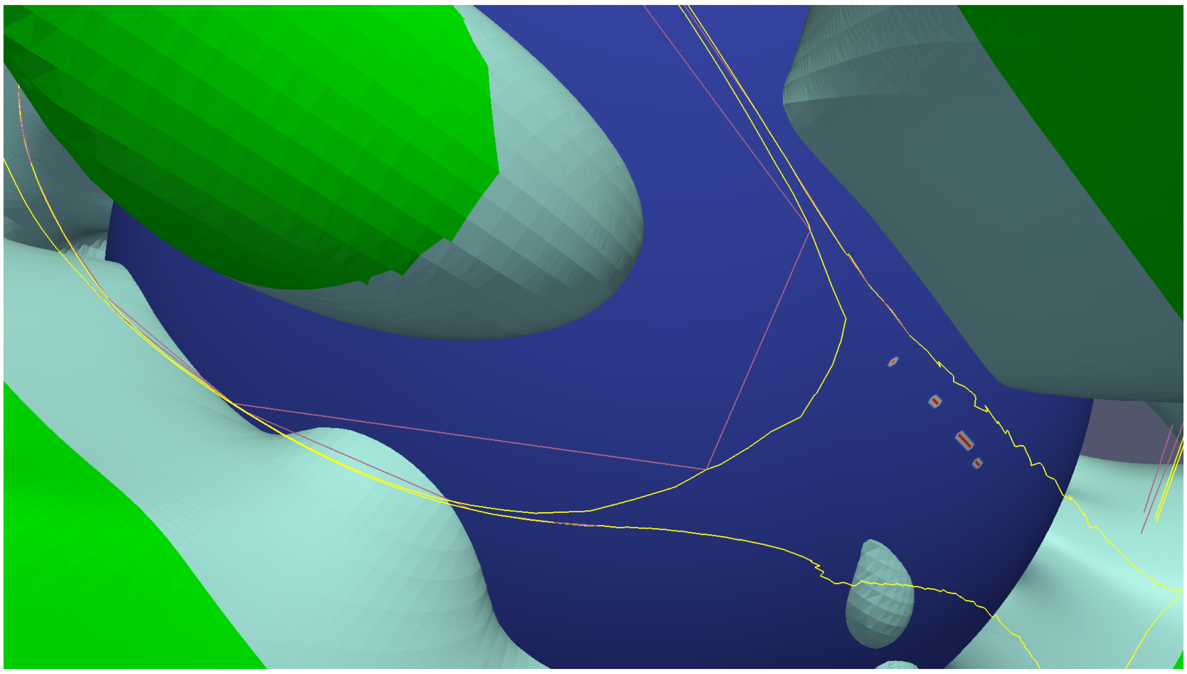

Figure 4.6.: Comparison between original CBL intersections (magenta) and the refinement of them (yellow).

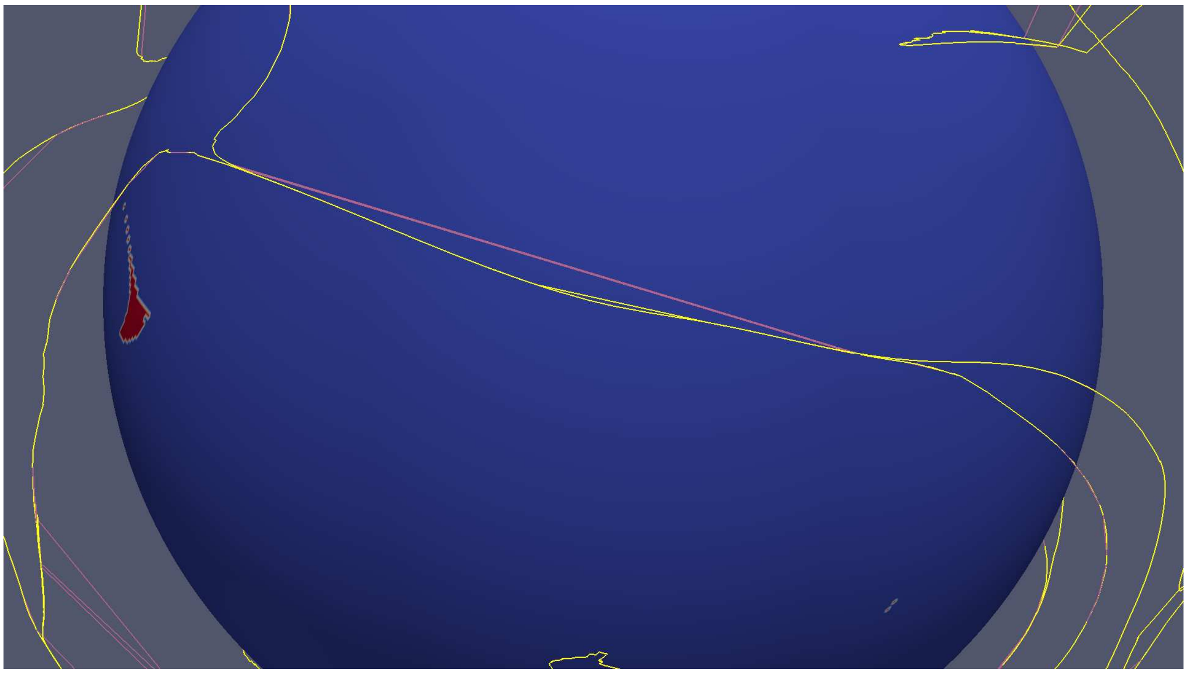

Figure 4.7.: A candidate for a corridor between neighbouring coronal holes.

5. Implementation

We make heavy use of templates in our program. Furthermore, we take advantage of new features added in C++11 (ISO/IEC 14882:2011)[1], including default template arguments, range-based `for` loops and variadic templates.

The following folders contain the code:

- `misc/`, containing all kind of useful generic classes like vectors
- `ReadMAS/`, providing functions for loading HDF files[2]
- `HDFExtract/`, a wrapper for ReadMAS
- `Policies/`, providing policiy classes
- `Info/`, for printing infos on the screen
- `GPU/`, containing the OpenCL streamtracer and
- `CoronalHoles/`, with the main program inside.

In order to minimize error-proneness and simplifying the task of adding new features to the code, many design patterns and templated classes are provided.

We use VTK[3] to visualize our data and to store it. At first we tried to use the streamtracer provided by VTK, but this streamtracer was not able to make use of the p-space to c-space conversion we used to accelerate the process of cell location. We also parallelized the streamtracer using pthreads, but it turned out that the grid had to be copied for every instance of the streamtracer since concurrent access to one grid resulted in segfaulting of the program. Furthermore, even with 8 instance of the streamtracer working in parallel, the reduction of the isosurface took more than 12 hours before we aborted the operation. All this lead to the need of a fast streamtracer, which was implemented using OpenCL. The streamtracer is only able to operate in uniform grids, and all seeds have to be transformed before passing them to the GPU.

All methods and classes are defined in a namespace, namely `CoronalHoles`

[1] `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372`
[2] Provided by Thomas Müller
[3] `http://www.vtk.org/`

5.1. Classes

ActorCreator	implemented in `misc/`
Provides methods for creating a variety of different VTK actors.	

Array	implemented in `misc/`
Encapsulates a simple C array, but is also capable of shallow and deep copying.	

BitSet	implemented in `misc/`
As the name implies, a simple bit set. Since the STL only provides bit sets where the number of elements must be known at compile time, this class was created.	

Buffer	implemented in `misc/`
Used to store earlier computed streamlines.	

CheckPolicies	implemented in `Policies/`
Class which enables the programmer to define different run-time checks for different cases.	

CLTracer	implemented in `GPU/`
Provides OpenCL functionality.	

CmdArgsParser	implemented in `misc/`
Maps command line arguments to function calls.	

CopyPolicies	implemented in `Policies/`
Defines if an object is either deep copied or only a shallow copy.	

CPUBuffer	implemented in `GPU/`
Simplifies the process of allocating memory on the CPU.	

Cube	implemented in `misc/`
Defines an arbitrary cube. The cubes vertices can contain both position and scalars/vectors. There is also a version which is able to share vertices with other cubes.	

Edge	implemented in `misc/`
A line with only two points. The order of the two points doesn't matter when two edges are compared. This is needed for edge extraction of the poly objects.	

Exception	implemented in `misc/`
Contains all exceptions in use.	

GeometryCreator	implemented in `misc/`
Responsible for constructing the grid, e.g., the unstaggering of the HDF data.	

GPUBuffer	implemented in `GPU/`
Simplifies the process of allocating memory on the GPU.	

GridHolder	implemented in `misc/`
Contains the grid and methods which simplify the usage of it.	

Grid	implemented in `misc/`
Stores regular grids.	

Gui	implemented in `misc/`
Interface between VTK, QT and the user.	

HDFExtract	implemented in `HDFExtract/`
Loads HDF files and access operators.	

Info	implemented in `Info/`
Contains code for an additional information window besides the console.	

InfoPrint	implemented in `Info/`
Enables output on the console as well as in a separate QT window.	

Interpolator	implemented in `misc/`
Provides methods for linear, bilinear and trilinear interpolation.	

LinearScaleUtility	implemented in `misc/`
Simply provides functions for mapping values from one range to another.	

Line	implemented in `misc/`
Contains a line, stored in a STL list for support of simple insertion and deletion of points without invalidating existing iterators.	

LogManager	implemented in `misc/`
If one wants to write a logfile, this class provides the necessary methods.	

Matrix	implemented in `misc/`
Generic class for matrices.	

MemoryAllocator	implemented in `misc/`
Simplifies dynamic memory allocation.	

Mesh	implemented in `misc/`
Triangle or Quad mesh. Supports edge extraction.	

Options	implemented in `misc/`
Contains the options needed by **ActorCreator**.	

Print	implemented in `misc/`
Provides different functions for printing text on the console.	

Prototypes	implemented in `misc/`
Contains prototypes of the classes.	

Quad	implemented in `misc/`
Similar to the **Triangle** class, just with four points instead of three.	

RenderWindow	implemented in `misc/`
Class that holds references to all options and VTK actors. Also responsible for user interaction.	

StreamTracer	implemented in `GPU/`
Performs stream tracing on the GPU.	

StructureCreator	implemented in `misc/`
Responsible for a variety of tasks. Creates streamlines, cuts polylines, adaptively refines structures, writes poly objects to files and many more.	

Surface	implemented in `misc/`
A special case of a mesh. It's a two-dimensional manifold in a three-dimensional space.	

Threader	implemented in `misc/`
Provides a way to define threads using pthreads.	

Timer	implemented in `misc/`
Enables time measurement.	

Triangle	implemented in `misc/`
A plane defined by three points. Able to share vertices with other triangles.	

VarWrap	implemented in `misc/`
Enables operations on C arrays which are well known from textures. Values outside the range of the array can be either lead to an exception, the index can be wrapped or clamped.	

Vector	implemented in `misc/`
Generic class for vectors.	

Vertex	implemented in `misc/`
Contains position and scalar/vector values.	

WrapPolicies	implemented in `Policies/`
Defines the policies for the class **VarWrap**.	

6. Results

6.1. Runtime Estimation

Since cell location is a trivial task in our case with constant time ($\mathcal{O}(1)$), our streamtracer scales lineary in terms of the number of seeds $|S|$ and propagation time p ($\mathcal{O}(|S| \cdot p)$).

Constructing the isosurface is linear in the number of cells $|C|$ ($\mathcal{O}(|C|)$).

The most expensive task is the reduction of the isosurfaces: Using marching cubes, a cell can contain up to 5 triangles. Thus, an isosurface passing n cells can consist of $n \cdot 5$ triangles. Except for the boundaries of the isosurface, the triangles are sharing their vertices reducing the total amount of vertices. In the end, the number of vertices of an isosurface lies in $\mathcal{O}(n)$. Assuming we have $|I|$ isosurfaces and exploiting the fact that no two isosurfaces can intersect, the number of vertices of all isosurfaces is $\mathcal{O}(|I| \cdot \frac{|C|}{|I|}) = \mathcal{O}(|C|)$. Since at every vertex, two streamlines have to be started, and removal of triangles is logarithmic in time due to use of hash maps, the complete task is in $\mathcal{O}(|C| \cdot \log(|C|))$

Mesh boundary extraction is in $\mathcal{O}(\log(|C|))$ since the number of edges is limited by the number of triangles, and the extraction of the boundaries is realized by utilizing hash maps. Insertion as well as removal is logarithmic in time when using hash maps.

Computing the intersection of the streamlines with a sphere is linear in the length of the streamlines, which is limited by the chosen propagation ($\mathcal{O}(p)$)

Adaptive refinement can also be expensive, but since only a few segments tend to be too long this is not a big issue. Let l_{limit} be the desired segment length on the photosphere. Adaptive refinement scales lineary with the number of mesh boundaries $|E|$. The maximum segment length which can occure in the dataset is determined by the diameter of the grid representation d, so there have to be at least $\lceil \frac{d}{l_{limit}} \rceil - 1$ new streamlines to be created and intersected. In the worst case, no segment length reduction is achieved, so that after a fixed limit l_{it} of iterations over the segment (usually 50), the operation is terminated. This leads to a runtime estimation of $\mathcal{O}(l_{it} \cdot \log(|C|) \cdot |S| \cdot p \cdot p) = \mathcal{O}(\log(|C|) \cdot |S| \cdot p^2) = \mathcal{O}(|C|)$.

This results in a total runtime of $\mathcal{O}(|C|) + \mathcal{O}(\log(|C|)) + \mathcal{O}(p) + \mathcal{O}(\log(|C|) \cdot |S| \cdot p^2) = \mathcal{O}(|C| + \log(|C|) + p + \log(|C|) \cdot |S| \cdot p^2)$

Summary

Structure	Number of elements
Grid cells	$\|C\| = 256 \cdot 150 \cdot 200 = 7680000$
Isosurface vertices	$\mathcal{O}(\|C\|)$
Isosurface triangles	$\mathcal{O}(\|C\|)$
Mesh Boundaries	$\mathcal{O}(\|I\|)$
Seeding points	$\mathcal{O}(\|I\|)$

Operation	Runtime
Cell location	$\mathcal{O}(1)$
Streamtracing	$\mathcal{O}(\|S\| \cdot p)$
Isosurface construction	$\mathcal{O}(\|C\|)$
Isosurface reduction	$\mathcal{O}(\|C\| \cdot \log(\|C\|))$
Mesh Boundary Extraction	$\mathcal{O}(\log(\|C\|))$
Streamline intersection	$\mathcal{O}(p)$
Adaptive refinement	$\mathcal{O}(\|C\|)$
Total runtime	$\mathcal{O}(\|C\|)$

Operation	**Time needed for computation of figures**[a] [a]AMD Phenom II X6 1055T Processor x6, 8 GB RAM, ATI Radeon 5850
Texture based coronal hole map (512x512) $\rightarrow$ 524288 streamlines	20m:34s
Isosurface generation	00m:03s
Reduction of isosurface $\rightarrow$ 254157 streamlines	11m:42s
Edge extraction	00m:07s
Adaptive refinement	03m:17s

6.2. Future Work

The topological division of the magnetic field in two distinct parts is an interesting feature of coronal hole boundaries and their corresponding CBLs. An investigation in extracting CBLs in another manner than by isosurface reduction is desirable and might make the adaptive refinement unnecessary. Pilipenko investigated relationships between different fields supplied of the Helioseismic and Magnetic Imager Solar Dynamics Observatory, including the relations between the magnetic field and the current density [Pil12]. There may be a way to utilize these insights to compute other topological features which are not apparent at the moment, and use the gained knowledge to improve the CBL evaluation.

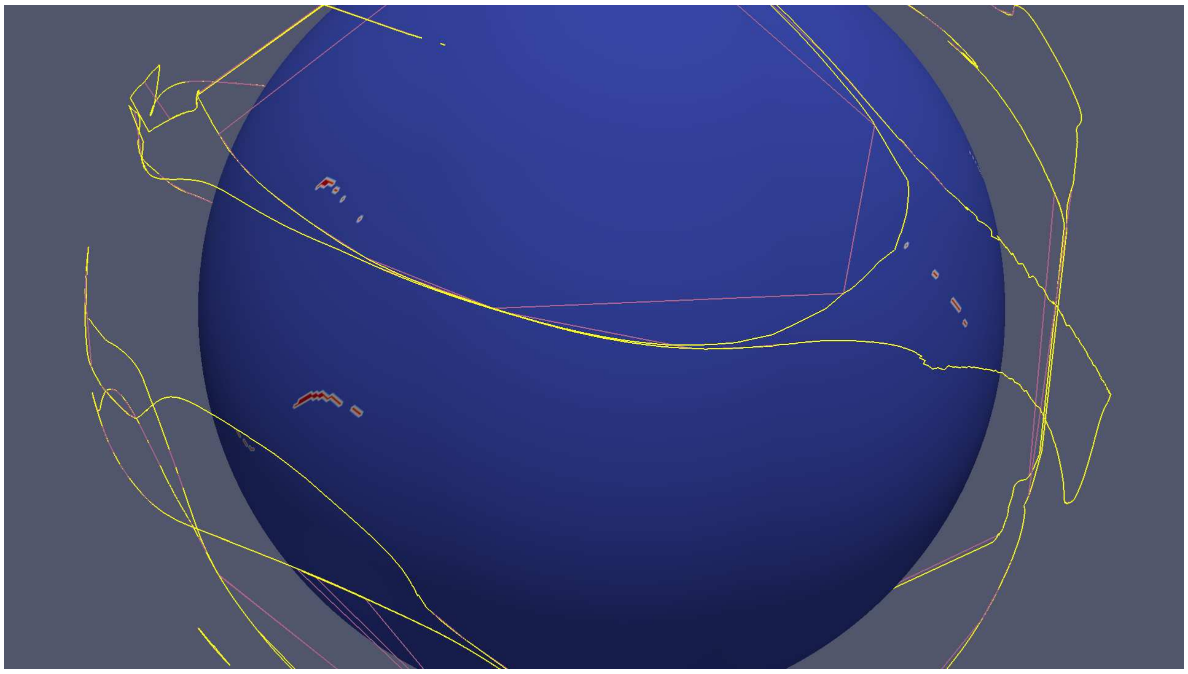

Figure 6.1.: Adaptive refinement with resulting coronal hole corridor.

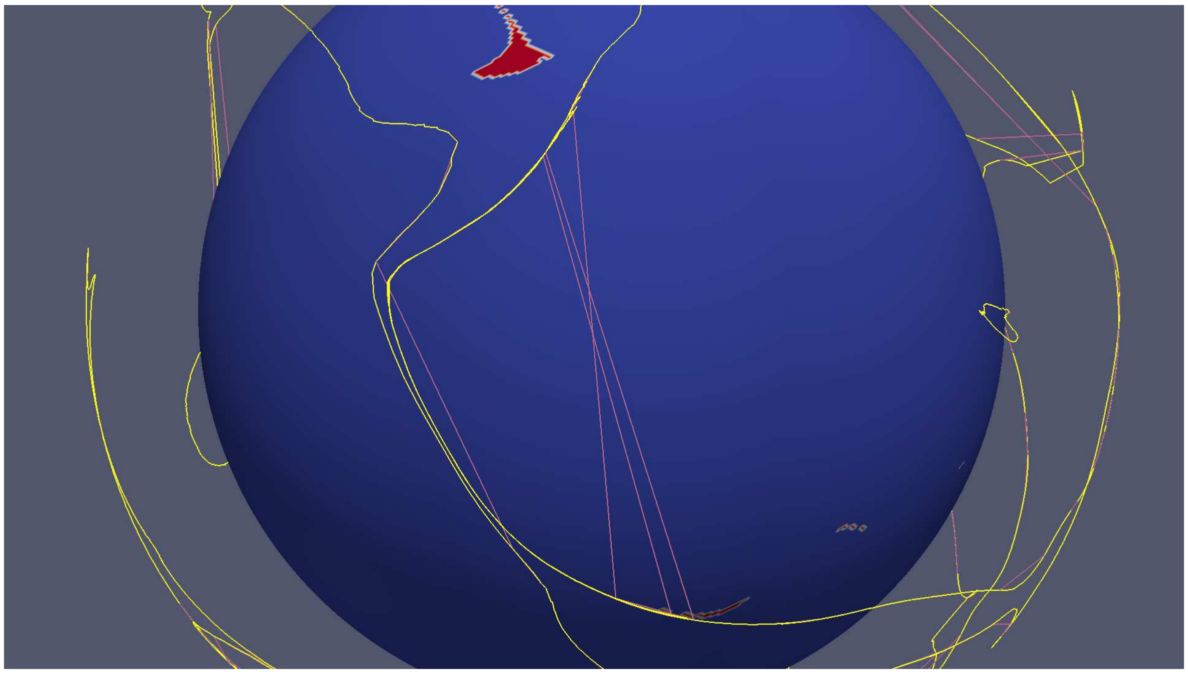

Figure 6.2.: Difference between unrefined (magenta) and refined (yellow) coronal hole boundaries.

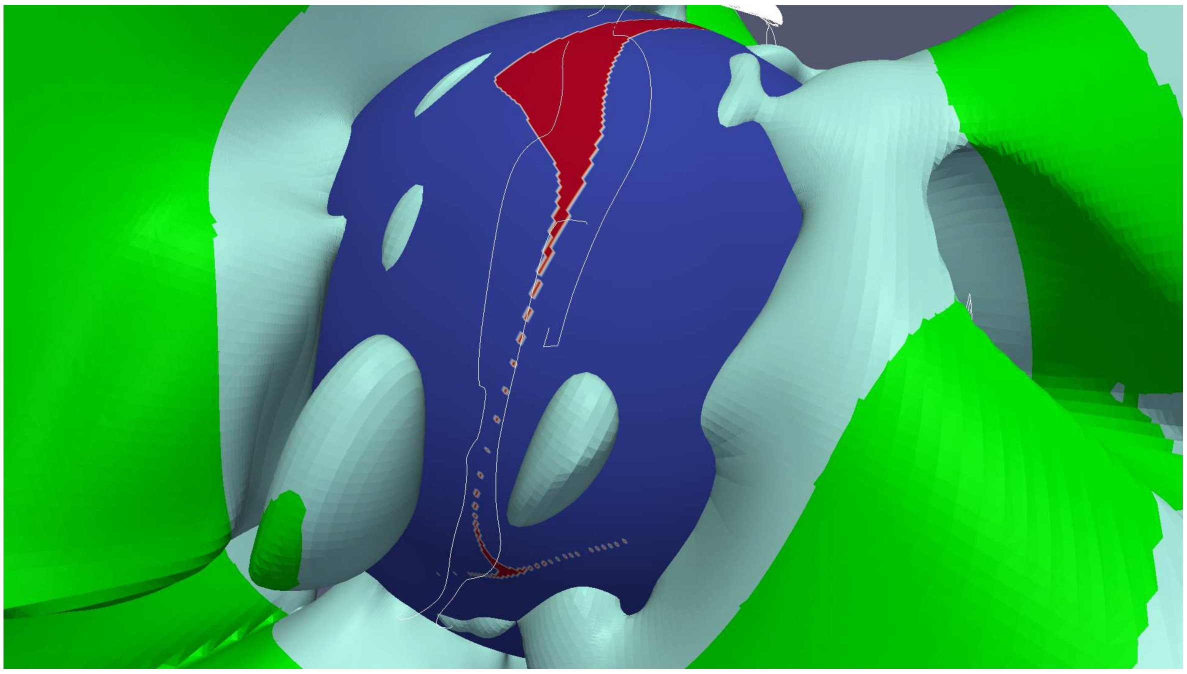

Figure 6.3.: Approach of extracting coronal hole boundaries with isolines of $\mathbf{B}_r = 0$ at sphere r_p instead of reduced isosurfaces. It resulted in segmentation of the coronal hole boundaries.

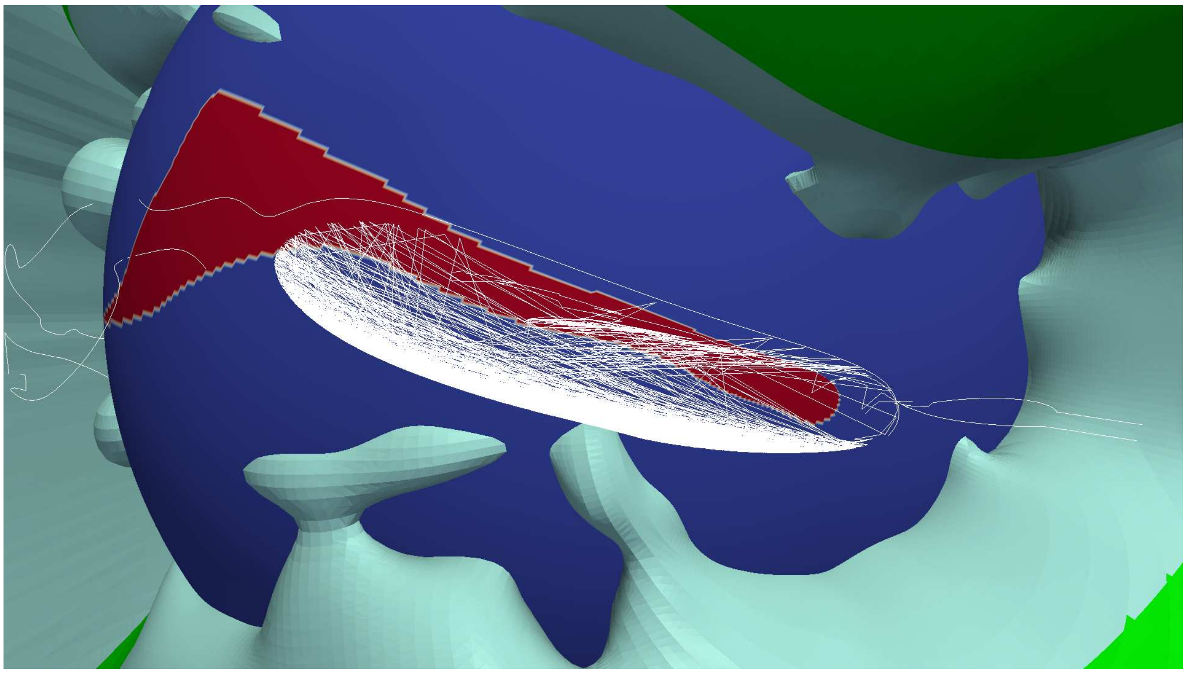

Figure 6.4.: Oscillating polyline result in isoline approach.

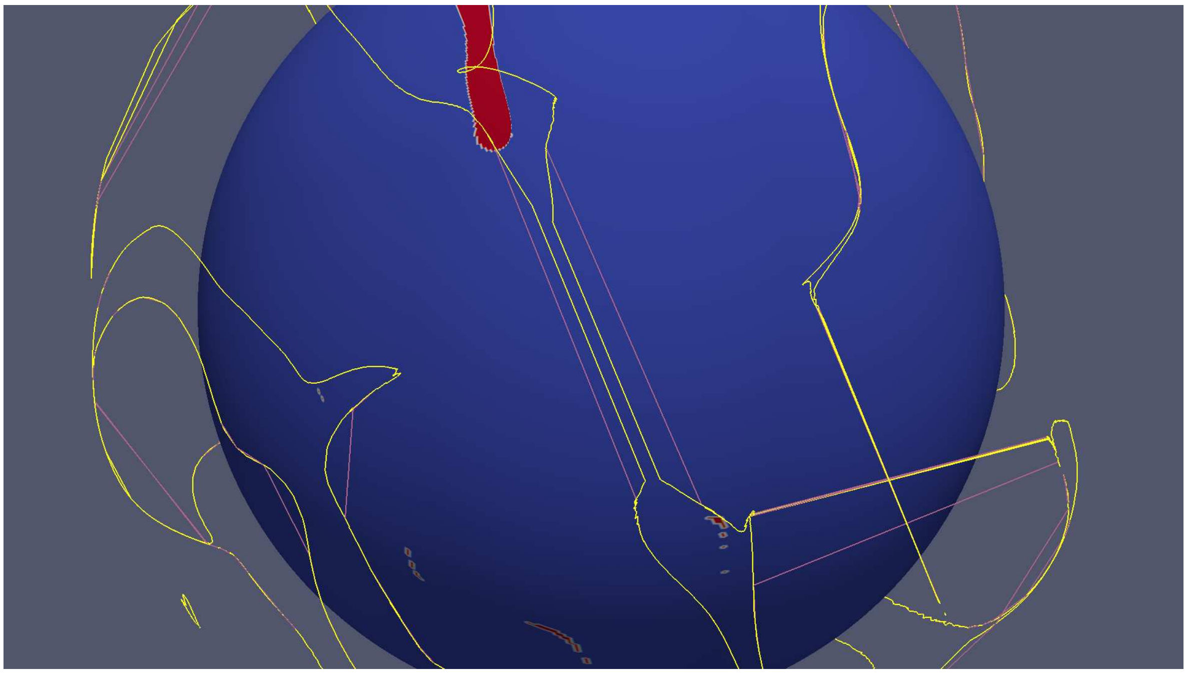

Figure 6.5.: Corridor which could not be refined until a sufficient representation was aquired.

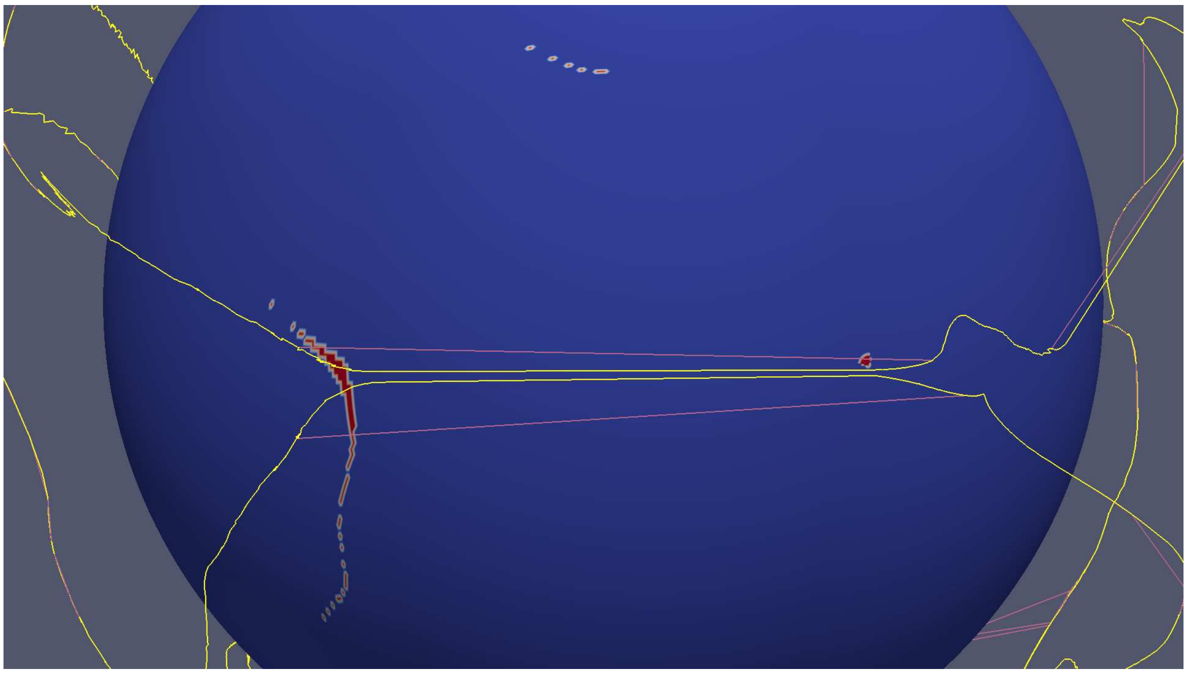

Figure 6.6.: Another corridor where the refinement failed but nethertheless the result is accurate enough for further refinement.

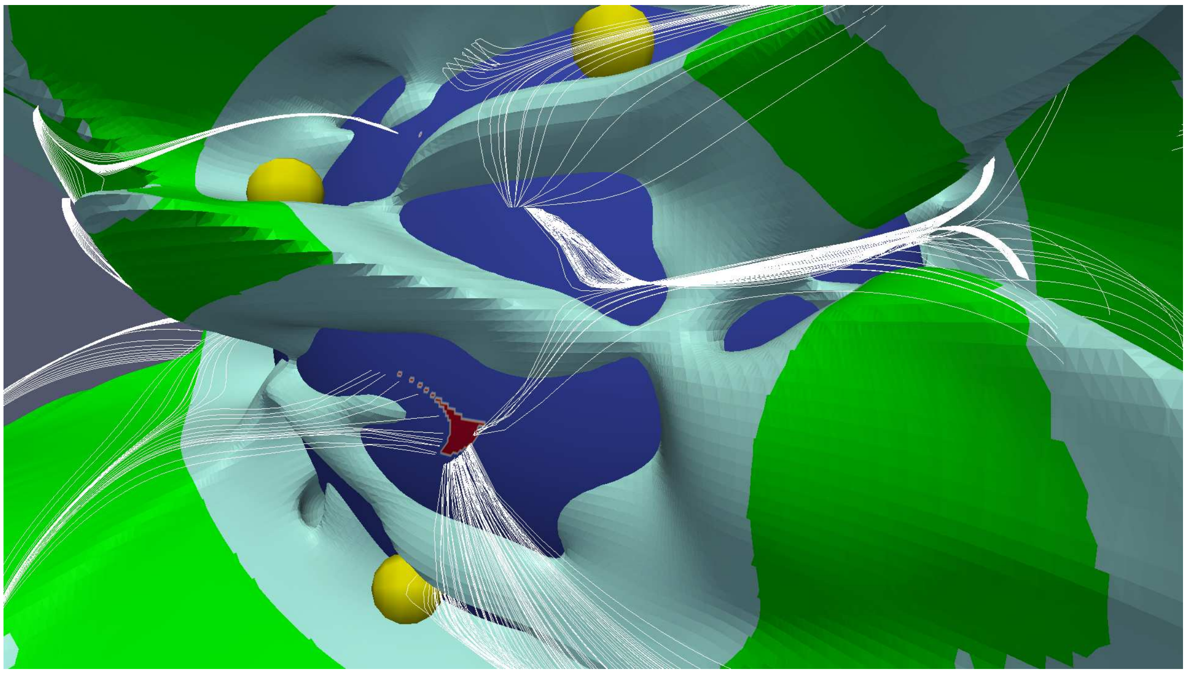
Figure 6.7.: Repelling of streamlines near yellow critical points.

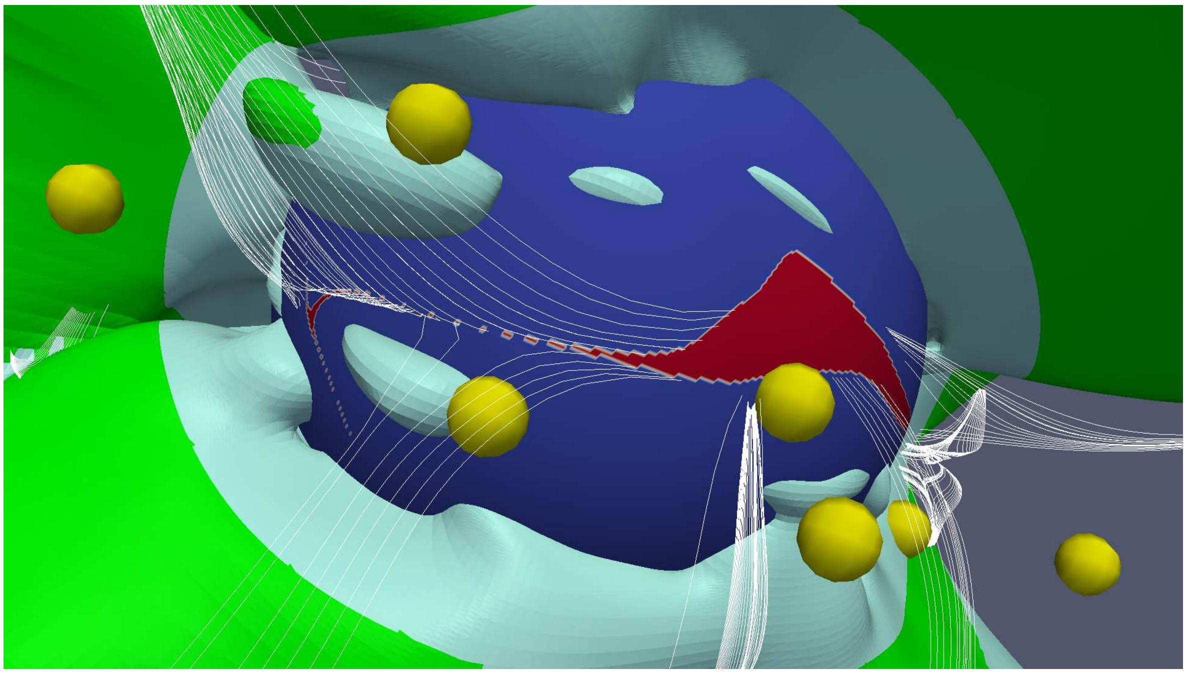
Figure 6.8.: Repelling of streamlines near yellow critical points.

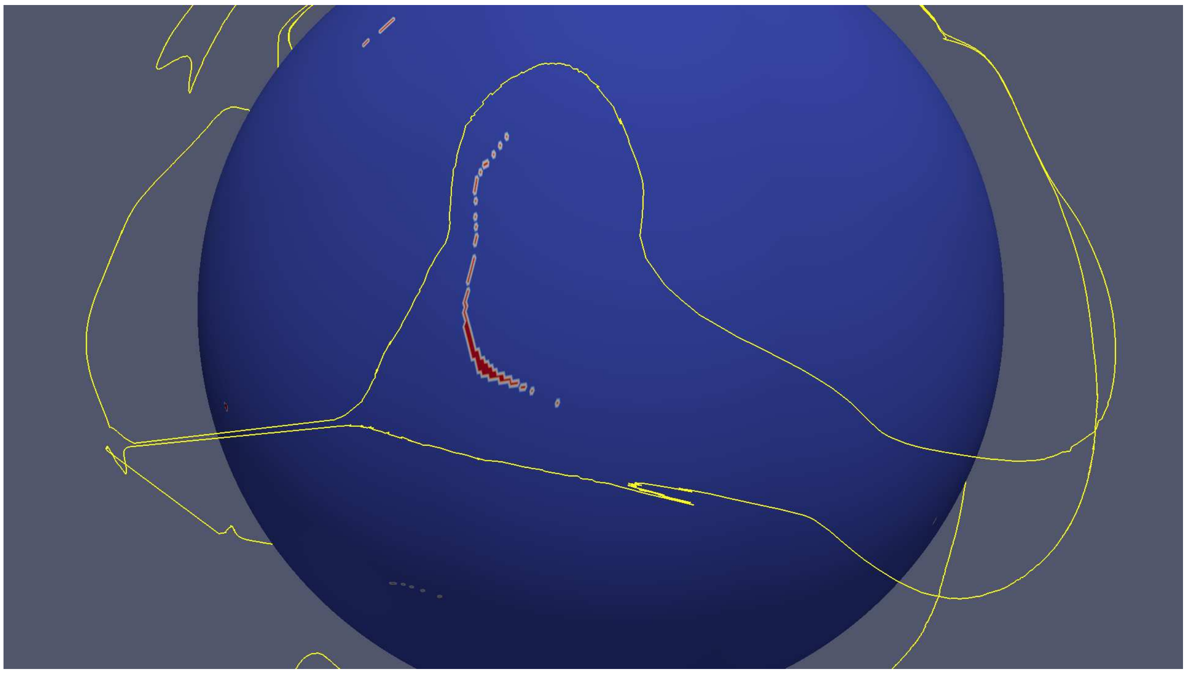

Figure 6.9.: Difference between texture based coronal hole rendering and coronal hole boundary extraction. Note the corridor at the left side of the image.

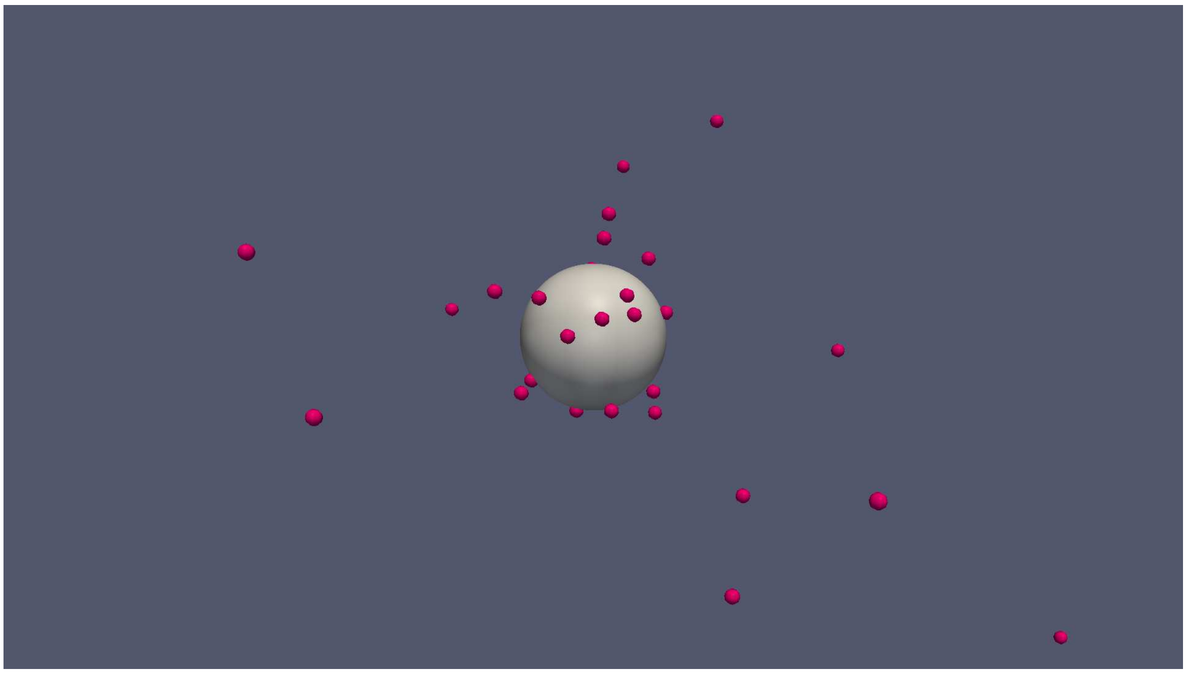

Figure 6.10.: Critical points (magenta) near the surface of the Sun.

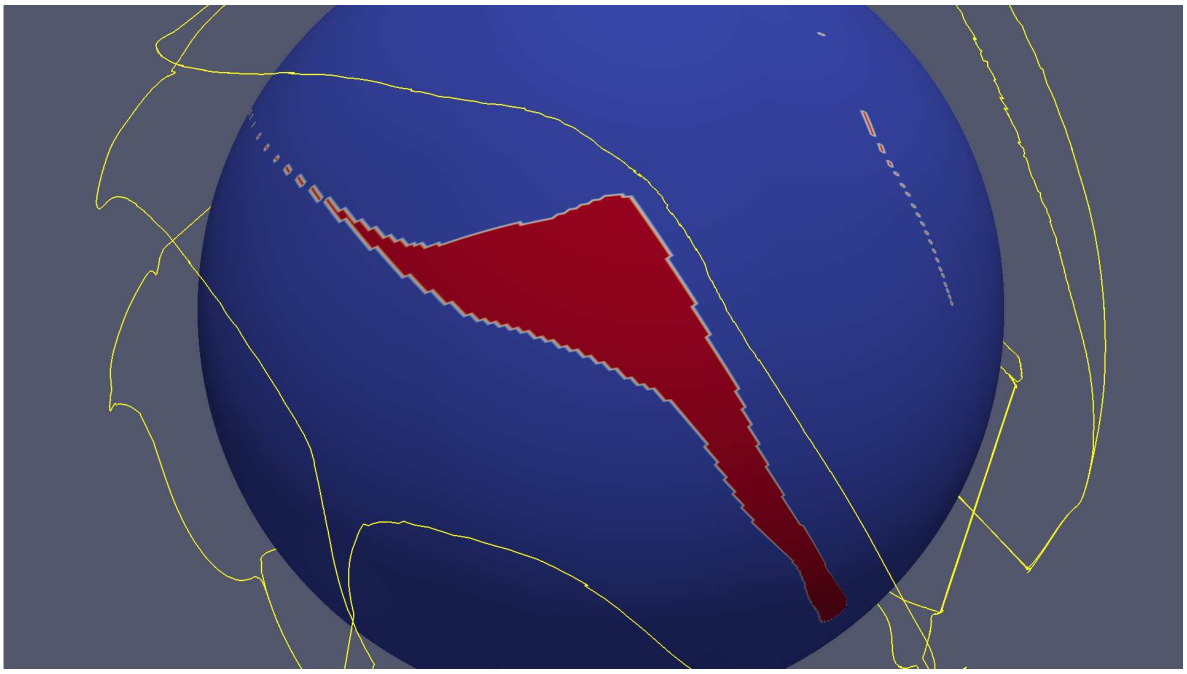

Figure 6.11.: Compliance of coronal hole boundary extraction and texture based coronal hole rendering.

Adaptive refinement is the greatest challenge of coronal boundary extraction due to the repelling character of critical points (figure 6.11). It may be possible to make use of critical points or bifurcation lines to simplify the process. It may also be possible to utilize the texture based method to gain additional information on how to decide on a good refinement strategy. Filtering the MHD data before processing to reduce the advent of critical points might also be helpful.

The next step for coronal hole boundaries extraction is to identify corridors and classify them in terms of connectivity. The topological evolution of both the coronal holes and the corridors is also of great interest. This could be achieved by using level sets to construct level surfaces. Analyzing them for repetitive patterns may lead to new insights on solar dynamics and help to forecast geomagnetic storms.

7. Algorithms

Algorithmus 7.1 Classical Fourth-Order Runge-Kutta Method

Input:

- Seeding point
- Grid
- Number of points to generate
- Step size
- Integration direction (forward/backward)

Output: A streamline starting at *seed*, consisting of *numPoints* points

```
function RungeKutta4($seed$, $grid$, $numPoints$, $stepSize$, $direction$)
    Line $l$
    Point $p \leftarrow seed$, $p_2$
    Cell $c$
    for $i = 1 \rightarrow numPoints$ do
        $appendPoint(l, p)$
        $c \leftarrow getCell(grid, p)$
        $k_1 \leftarrow interpolateTrilinear(c, p)$
        if $direction = FORWARD$ then
            $p_2 \leftarrow p + k_1 \cdot \frac{stepSize}{2}$
        else
            $p_2 \leftarrow p - k_1 \cdot \frac{stepSize}{2}$
        end if
        $k_2 \leftarrow interpolateTrilinear(c, p_2)$
        if $direction = FORWARD$ then
            $p_2 \leftarrow p + k_2 \cdot \frac{stepSize}{2}$
        else
            $p_2 \leftarrow p - k_2 \cdot \frac{stepSize}{2}$
        end if
        $k_3 \leftarrow interpolateTrilinear(c, p_2)$
        if $direction = FORWARD$ then
            $p_2 \leftarrow p + k_3 \cdot stepSize$
        else
            $p_2 \leftarrow p - k_3 \cdot stepSize$
        end if
        $k_4 \leftarrow interpolateTrilinear(c, p_2)$
        if $direction = FORWARD$ then
            $p_2 \leftarrow p + (k_1 + 2k_2 + 2k_3 + k4) \cdot \frac{stepSize}{6}$
        else
            $p_2 \leftarrow p - (k_1 + 2k_2 + 2k_3 + k4) \cdot \frac{stepSize}{6}$
        end if
        $p \leftarrow p_2$
    end for
    return $l$
end function
```

Algorithmus 7.2 Index ϕ, θ, r to Spherical Coordinates

```
Input: Indices $\phi \in \mathbb{N}, \theta \in \mathbb{N}$, radius $r \in \mathbb{R}^+$ and the corresponding maximal values of these
  indices $max_\phi, max_\theta$
Output: Spherical coordinates for the given values
  function INDICESTOSPHERICALCOORDS($\phi, \theta, r, max_\phi, max_\theta$)
      $\vec{p} \leftarrow \vec{0}$
      $\Phi \leftarrow \frac{mod(\phi, max_\phi)}{max_\phi - 1} \cdot 2\pi$
      $\Theta \leftarrow \frac{mod(\theta, max_\theta)}{max_\theta - 1} \cdot \pi$
      $\vec{p_x} \leftarrow r \cdot \sin(\Theta) \cdot \cos(\Phi)$
      $\vec{p_y} \leftarrow r \cdot \sin(\Theta) \cdot \sin(\Phi)$
      $\vec{p_z} \leftarrow r \cdot \cos(\Theta)$
      return $\vec{p}$
  end function
```

Algorithmus 7.3 Index ϕ, θ, r to Sphere Gradient

```
Input: Indices $\phi \in \mathbb{N}, \theta \in \mathbb{N}$, radius $r \in \mathbb{R}^+$ and the corresponding maximal values of these
  indices $max_\phi, max_\theta$
Output: Sphere gradient for the given values
  function INDICESTOSPHEREGRADIENT($\phi, \theta, r, max_\phi, max_\theta$)
      $\Phi \leftarrow \frac{mod(\phi, max_\phi)}{max_\phi - 1} \cdot 2\pi$
      $\Theta \leftarrow \frac{mod(\theta, max_\theta)}{max_\theta - 1} \cdot \pi$
      return $\begin{pmatrix} -r \cdot \sin(\Theta)\sin(\Phi) & r \cdot \cos(\Theta)\cos(\Phi) & \sin(\Theta)\cos(\Phi) \\ r \cdot \sin(\Theta)\cos(\Phi) & r \cdot \cos(\Theta)\sin(\Phi) & \sin(\Theta)\sin(\Phi) \\ 0 & -r \cdot \sin(\Theta) & \cos(\Theta) \end{pmatrix}$
  end function
```

Algorithmus 7.4 Staggered Grid to Curvilinear Grid

```
Input: HDF data files
Output: The resulting grid
  function STAGGEREDGRIDTOCURVILINEARGRID($hdfFiles[]$)
      $B_\phi[][][] \leftarrow hdfFiles[0]$
      $B_\theta[][][] \leftarrow hdfFiles[1]$
      $B_r[][][] \leftarrow hdfFiles[2]$
      $grid[B_\phi.size(\phi)][B_\theta.size(\theta)][B_r.size(r)]$
      for $r = 0 \rightarrow B_r.size(r) - 1$ do
          for $\theta = 0 \rightarrow B_\theta.size(\theta) - 1$ do
              for $\phi = 0 \rightarrow B_\phi.size(\phi) - 1$ do
                  $grid[\phi][\theta][r].pos \leftarrow$
                      $indicesToSphericalCoords(\phi, \theta, B_r[\phi][\theta][r], B_\phi.size(\phi), B_\theta.size(\theta))$
                  $grid[\phi][\theta][r].val_x \leftarrow interpolateLinear(B_\phi[\phi][\theta][r], B_\phi[\phi+1][\theta][r], 0.5)$
                  $grid[\phi][\theta][r].val_y \leftarrow interpolateLinear(B_\theta[\phi][\theta][r], B_\theta[\phi][\theta+1][r], 0.5)$
                  $grid[\phi][\theta][r].val_z \leftarrow interpolateLinear(B_r[\phi][\theta][r], B_r[\phi][\theta][r+1], 0.5)$
                  $\mathbf{m} \leftarrow indicesToSphereGradient(\phi, \theta, B_r[\phi][\theta][r], B_\phi.size(\phi), B_\theta.size(\theta))$
                  $grid[\phi][\theta][r].val \leftarrow \mathbf{m} \cdot grid[\phi][\theta][r].val$
              end for
          end for
      end for
      return $grid$
  end function
```

Algorithmus 7.5 Isosurface Removal

```
Input: The isosurface and the radius which classifies a streamline to be contained in the
  magnetic field or not.
Output: The reduced isosurface
  function REDUCEISOSURFACE($mesh, radius$)
      for $i = 0 \rightarrow mesh.vertices.size() - 1$ do
          Line $l[2] \leftarrow startStreamlineInBothDirections(mesh.vertices[i])$
          if $l[0] \cap Sphere(radius) \neq \emptyset$ AND $l[1] \cap Sphere(radius) \neq \emptyset$ then
              $mesh.removeVertexAndNeighbouringTriangles(mesh.vertices[i])$
          end if
      end for
      return $mesh$
  end function
```

Algorithmus 7.6 Edge Detection

```
Input: A triangle mesh
Output: Edges which belong to only one triangle
  function GETEDGES($triangles, numTriangles$)
      HashMap $edges$
      HashMap $tempEdges$
      for $i = 0 \to numTriangles - 1$ do
          Point $p[3] \leftarrow triangles[i].getVertices()$
          for $j = 0 \to 2$ do
              Edge $e \leftarrow$ Edge($p[j], p[mod(j + 1, 3)]$)
              $edges.insert(e)$
              if $tempEdges.find(e)$ then
                  $edges.erase(e)$
              else
                  $tempEdges.insert(e)$
              end if
          end for
      end for
      List $result$
      for $i = 0 \to edges.size() - 1$ do
          $result.insert(edges[i])$
      end for
      return $result$
  end function
```

Algorithmus 7.7 Merge Edge Segments

```
Input: A list of edges
Output: Lines which contain the merged edges
  function MERGEEDGESEGMENTS($edges$)
      List $lines$
      for $i = 0 \to edges.size() - 1$ do
          Bool $vertexFound \leftarrow$ FALSE
          for $j = 0 \to lines.size()$ do
              if $lines[j].first = edges[i].first)$ then
                  $vertexFound \leftarrow$ TRUE
                  $lines[j].prepend(edges[i].second)$
                  BREAK
              else if $lines[j].second = edges[i].first)$ then
                  $vertexFound \leftarrow$ TRUE
                  $lines[j].append(edges[i].second)$
                  BREAK
              else if $lines[j].first = edges[i].second)$ then
                  $vertexFound \leftarrow$ TRUE
                  $lines[j].prepend(edges[i].first)$
                  BREAK
              else if $lines[j].second = edges[i].second)$ then
                  $vertexFound \leftarrow$ TRUE
                  $lines[j].append(edges[i].first)$
                  BREAK
              end if
          end for
          if $vertexFound$ then
              CONTINUE
          else
              $lines.insert(Line(edges[i].first, edges[i].second))$
          end if
      end for
      $lines \leftarrow mergeLines(lines)$
      return $lines$
  end function
```

Algorithmus 7.8 Merge Lines

Input: A list of lines

Output: The list of merged lines

```
  function mergeLines($lines$)
      Bool $linesChanged$
      repeat
          $linesChanged \leftarrow$ FALSE
          for $i = 0 \rightarrow lines.size() - 1$ do
              for $j = 1 \rightarrow lines.size() - 1$ do
                  if $lines[i].first = lines[j].first$ then
                      $linesChanged \leftarrow$ TRUE
                      $lines[i].prependReverse(lines[j])$
                      $lines.remove(j)$
                      $j \leftarrow j - 1$
                  else if $lines[i].first = lines[j].second$ then
                      $linesChanged \leftarrow$ TRUE
                      $lines[i].prepend(lines[j])$
                      $lines.remove(j)$
                      $j \leftarrow j - 1$
                  else if $lines[i].second = lines[j].first$ then
                      $linesChanged \leftarrow$ TRUE
                      $lines[i].append(lines[j])$
                      $lines.remove(j)$
                      $j \leftarrow j - 1$
                  else if $lines[i].second = lines[j].second$ then
                      $linesChanged \leftarrow$ TRUE
                      $lines[i].appendReverse(lines[j])$
                      $lines.remove(j)$
                      $j \leftarrow j - 1$
                  end if
              end for
          end for
      until $linesChanged =$ FALSE
      return $lines$
  end function
```

Algorithmus 7.9 Adaptive Refinement

Input: Mesh boundaries, the desired length limit and the radius r_p for the cut

Output: The adaptively refined coronal hole boundaries

```
function ADAPTIVEREFINEMENT($meshBoundaries, lengthLimit, radius$)
    List $lines \leftarrow meshBoundaries.getEdges()$
    List $tempLines$
    for $i = 0 \rightarrow lines.size() - 1$ do
        if $lines[i].length() < lengthLimit$ then
            List $seeds = createPointsInBetween(lines[i])$
            $tempLines = streamTraceAndCutRecursively(seeds, radius)$
        end if
    end for
    return $tempLines$
end function
```

8. Conclusion

Extracting coronal holes from the magnetic field data of the sun remains a difficult problem. With our approach, the extraction was possible, but it needed heavy computation and still has some flaws. Using our first approach based on isolines at a specific radius as seeding regions was not sufficient and leaded to segmented and incorrect coronal hole boundaries (figure 6.3 and 6.4). Our subsequent approach based on isosurfaces of $\mathbf{B}_r = 0$ was successful.

The biggest problem during extraction seems to be the fact that there are many critical points near the surface of the sun, leading to aberration of the streamlines and the need of very fine refinement (see figure 1.2 and 6.9). But even with many subdivisions, it was not possible to refine all segments correctly. The sawtooth patterns resulting from the coarse mesh boundary representation could not be resolved in a satisfactory manner. Subdividing the cells containing the mesh boundaries even further and applying the marching cubes scheme on them might be of help. Alternatively, one could make a brute force attempt and take these cells to sample them with enough points and start streamlines from there.

We were able to identify structures which resemble corridors between coronal holes, providing a basis for future work. However refinement of these corridors was only partially successful. There might be a correlation between these corridors and critical points, but this has yet to be confirmed.

A. Appendix

A.1. Manual

A.1.1. Compiling the Program

In order to compile the provided software, there are several prerequirements. The following dependencies must be met:

- CMake (version >= 2.8)
- VTK libraries (version >= 5.10)[1]
- QT libraries (version >= 4.8.3)
- pthread libraries
- OpenGL libraries[2]
- OpenCL libraries[3]
- HDF4 libraries (version >= 4.2.7)
- libjpeg
- libz

For compiling the main program, a sh-script[4] is provided. It first generates the doxygen documentation. Next, it compiles the libraries *ReadMAS*, *HDFExtract* and *clTracer*. After that, the main program *coronalHoles* is compiled.

[1]VTK needs to be compiled with QT support
[2]GL, GLU (and glfw for the test modules)
[3]The OpenCL C++ Wrapper is also needed
[4]*compile.sh*

A.1.2. Command Line Arguments

Command	**Effect**
- -*num-threads*	Number of cores to be used
- -*coronal-holes-sphere*	Display coronal holes using the standard method
- -*critical-points*	Display critical points
- -*no-critical-points*	Disable critical points calculation
- -*shrink*	Cells will be shrinked (you should shrink apx. 10 * (RAM size in GiB) / 2 spheres at max)

continued on next page

continued from previous page

Command	**Effect**
- -*tubes*	Create tubed mesh
- -*field*	Draw arrows which illustrate the underlying vector field
- -*isolines*	Create isolines
- -*threshold*	Use a specific value for thresholding
- -*isomin*	Which isoline to begin with (typically 0 - 2)
- -*isomax*	Which isoline to end with (typically 0 - 2)
- -*isoradmin*	Which isoline to begin with
- -*isoradmax*	Which isoline to end with
- -*separated-isolines*	If isolines are created, they will be separated and not be one mesh
- -*gradients*	Create gradients
- -*sphere*	Render a sphere
- -*no-sphere*	Don't render the default sphere
- -*streamlines*	Create streamlines
- -*streamtubes*	Create streamtubes
- -*tube-radius*	Specify tube radius (default: 0.01)
- -*streamribbons*	Create streamribbons
- -*streamlines-from-isolines*	Create streamlines from isolines
- -*seedmin*	On which sphere to start the seedpoints for streamlines (needs to be in range of rmin and rmax)
- -*seedmax*	On which sphere to end the seedpoints for streamlines (needs to be in range of rmin and rmax)
- -*propagation*	Used for integration of streamlines
- -*runge-kutta*	Defines which integration method to use (2, 4, 45)
- -*cutter*	Make a cut through the sphere
- -*streamlines-cutter*	Make a cut through the streamlines
- -*streamlines-from-isolines-cutter*	Make a cut through the streamlines of isolines
- -*cutter-radius*	Define the radius of the cutting sphere
- -*streamline-limit*	Limit the number of streamlines which are computed in parallel. Useful when getting clOutOfResources exceptions
- -*write-to-file*	Write the data generated to files

Table A.1.: List of command line arguments

Bibliography

[Alf47] H. Alfvén. Magneto hydrodynamic waves, and the heating of the solar corona. *mnras*, 107:211, 1947. (Cited on page 10)

[BGT11] D. Banerjee, G. Gupta, L. Teriaca. Propagating MHD Waves in Coronal Holes. *Space Science Reviews*, 158:267–288, 2011. doi:10.1007/s11214-010-9698-z. URL http://dx.doi.org/10.1007/s11214-010-9698-z. (Cited on page 17)

[Cha07] J. Chae. Chromospheric Magnetic Reconnection. In K. Shibata, S. Nagata, T. Sakurai, editors, *New Solar Physics with Solar-B Mission*, volume 369 of *Astronomical Society of the Pacific Conference Series*, p. 243. 2007. (Cited on page 10)

[Cra09] S. R. Cranmer. Coronal Holes. *Living Reviews in Solar Physics*, 6(3), 2009. URL http://www.livingreviews.org/lrsp-2009-3. (Cited on page 17)

[GFT$^+$12] G.M. Machado, F. Sadlo, T. Müller, D. Müller, T. Ertl. Visualizing Solar Dynamics Data. *VMV 2012*, 2012. (Cited on page 17)

[Hea77] A. Hearn. An explanation of the observed differences between coronal holes and quiet coronal regions. *Solar Physics*, 51:159–168, 1977. doi:10.1007/BF00240453. URL http://dx.doi.org/10.1007/BF00240453. (Cited on page 17)

[KTR73] A. S. Krieger, A. F. Timothy, E. C. Roelof. A Coronal Hole and Its Identification as the Source of a High Velocity Solar Wind Stream. *solphys*, 29:505–525, 1973. doi:10.1007/BF00150828. (Cited on page 17)

[LC87] W. E. Lorensen, H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. doi: 10.1145/37402.37422. URL http://doi.acm.org/10.1145/37402.37422. (Cited on page 23)

[MC02] P. Martens, D. Cauffman. *Multi-Wavelength Observations of Coronal Structure and Dynamics*. Cospar Colloquia Series. Elsevier Science, 2002. URL http://books.google.de/books?id=wJ6xmA-ZpegC. (Cited on page 17)

[MW09] Madjarska, M. S., Wiegelmann, T. Coronal hole boundaries evolution at small scales. *A&A*, 503(3):991–997, 2009. doi:10.1051/0004-6361/200912066. URL http://dx.doi.org/10.1051/0004-6361/200912066. (Cited on page 17)

[MZO$^+$99] W. H. Matthaeus, G. P. Zank, S. Oughton, D. J. Mullan, P. Dmitruk. Coronal Heating by Magnetohydrodynamic Turbulence Driven by Reflected Low-Frequency Waves. *The Astrophysical Journal Letters*, 523(1):L93, 1999. URL http://stacks.iop.org/1538-4357/523/i=1/a=L93. (Cited on page 10)

[Pil12] I. Pilipenko. *Visualisierung magnetohydrodynamischer Felder*. Master's thesis, University of Stuttgart, Germany, 2012. (Cited on page 38)

[SH09] I. Scholl, S. Habbal. Automatic Detection and Classification of Coronal Holes and Filaments Based on EUV and Magnetogram Observations of the Solar Disk. In J. Ireland, C. Young, editors, *Solar Image Analysis and Visualization*, pp. 215–229. Springer New York, 2009. doi:10.1007/978-0-387-98154-3_16. URL http://dx.doi.org/10.1007/978-0-387-98154-3_16. (Cited on page 17)

[WTHS04] T. Weinkauf, H. Theisel, H.-C. Hege, H.-P. Seidel. Boundary Switch Connectors for Topological Visualization of Complex 3D Vector Fields. In *Proc. Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym '04)*, pp. 183–192. 2004. (Cited on page 12)

[YZJ+09] Yang, S. H., Zhang, J., Jin, C. L., Li, L. P., Duan, H. Y. Response of the solar atmosphere to magnetic field evolution in a coronal hole region. *A&A*, 501(2):745–753, 2009. doi:10.1051/0004-6361/200810601. URL http://dx.doi.org/10.1051/0004-6361/200810601. (Cited on page 17)

[ZWS+03] J. Zhang, J. Woch, S. K. Solanki, R. von Steiger, R. Forsyth. Interplanetary and solar surface properties of coronal holes observed during solar maximum. *J. Geophys. Res.*, 108(A4):1144–, 2003. URL http://dx.doi.org/10.1029/2002JA009538. (Cited on page 17)

All links were last followed on November 03, 2012.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Sebastian Jähne)